

REST \geq CRUD

Stefan Tilkov | innoQ | stefan.tilkov@innoq.com
<http://www.innoq.com> <http://railsconsulting.de>

Need Project Help?

WSDL
Linux
REST
Solaris
Hibernate
WS-*
Apache
Ruby Rails
JPA
SOAP
JRuby
Java
J2EE
Java EE
JSF

40 Developers – 5-10 on Rails

<mailto:info@innoq.com>

Audience Poll

How many of you make money doing Rails?

Percentage of Rails users developing RESTfully?

How many are just learning Ruby/Rails?

How many want to learn what REST is about?

How many know REST and want to see where I'm wrong?

What is REST?

3 definitions

1

REST: An Architectural Style

One of a number of “architectural styles”

... described by Roy Fielding in his dissertation

... defined via a set of *constraints* that have to be met

... architectural principles underlying HTTP, defined *a posteriori*

... with the Web as one particular instance

See: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

2

REST: The Web Used Correctly

A system or application architecture

... that uses HTTP, URI and other Web standards “correctly”

... is “on” the Web, not tunneled through it

... also called “WOA”, “ROA”, “RESTful HTTP”

3

REST: XML without SOAP

Send plain XML (w/o a SOAP Envelope) via
HTTP

- ... violating the Web as much as WS-*
- ... preferably use GET to invoke methods
- ... or tunnel everything through POST
- ... commonly called “POX”

***Only* option 1 is the right
one
(because Roy said so)**

**But we'll go with option 2
(and equate "REST" with
"RESTful HTTP usage")**

**and avoid option 3 like the
plague**

REST Explained in 5 Easy Steps

1. Give Every “Thing” an ID

`http://example.com/customers/1234`

`http://example.com/orders/2007/10/776654`

`http://example.com/products/4554`

`http://example.com/processes/sal-increase-234`

2. Link Things To Each Other

```
<order self='http://example.com/orders/1234'>  
  <amount>23</amount>  
  <product ref='http://example.com/products/4554' />  
  <customer ref='http://example.com/customers/1234' />  
</order>
```

3. Use Standard Methods

GET	retrieve information, possibly cached
PUT	Update or create with known ID
POST	Create or append sub-resource
DELETE	(Logically) remove

4. Allow for Multiple “Representations”

GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml

<customer>...</customer>

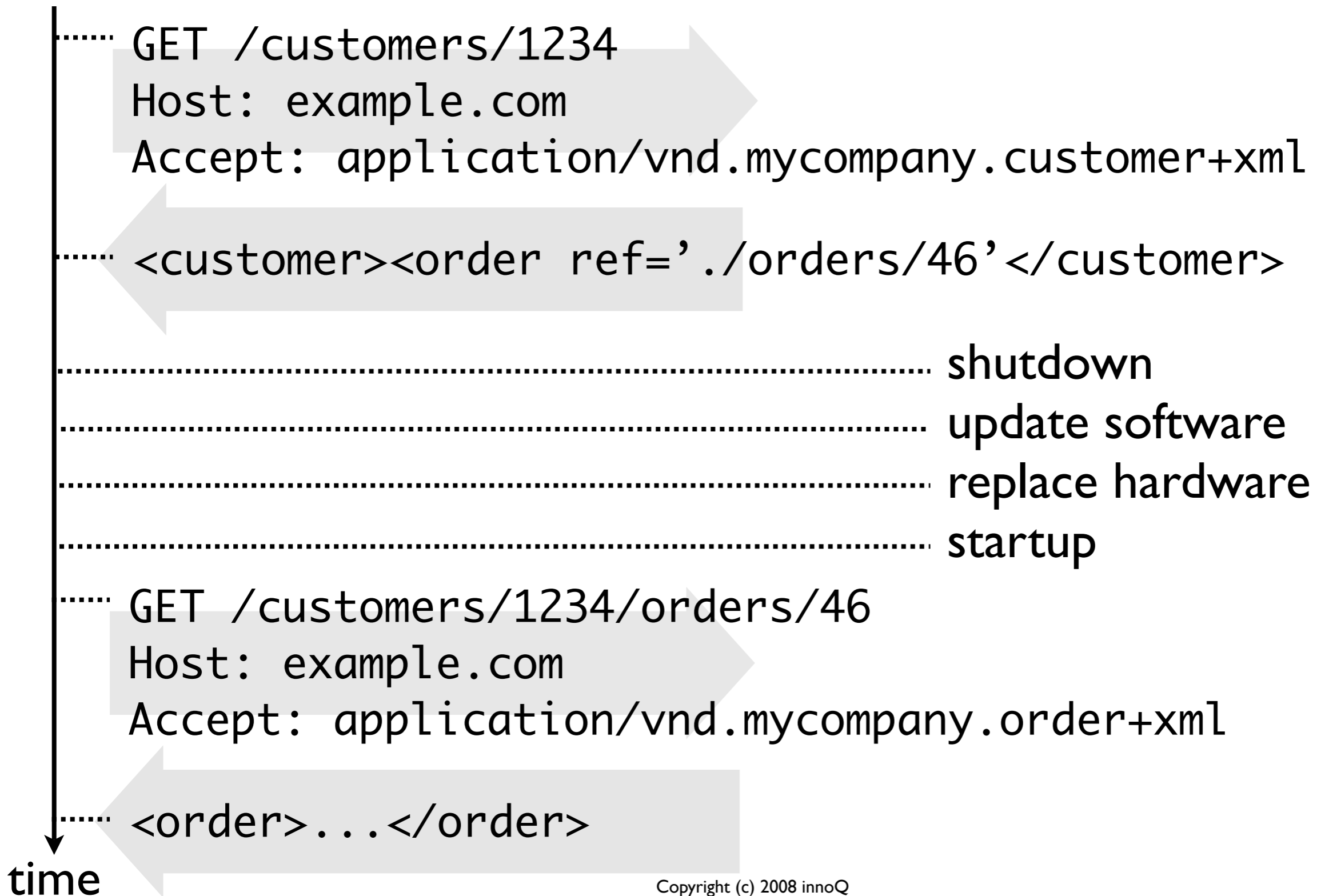
GET /customers/1234
Host: example.com
Accept: text/x-vcard

begin:vcard

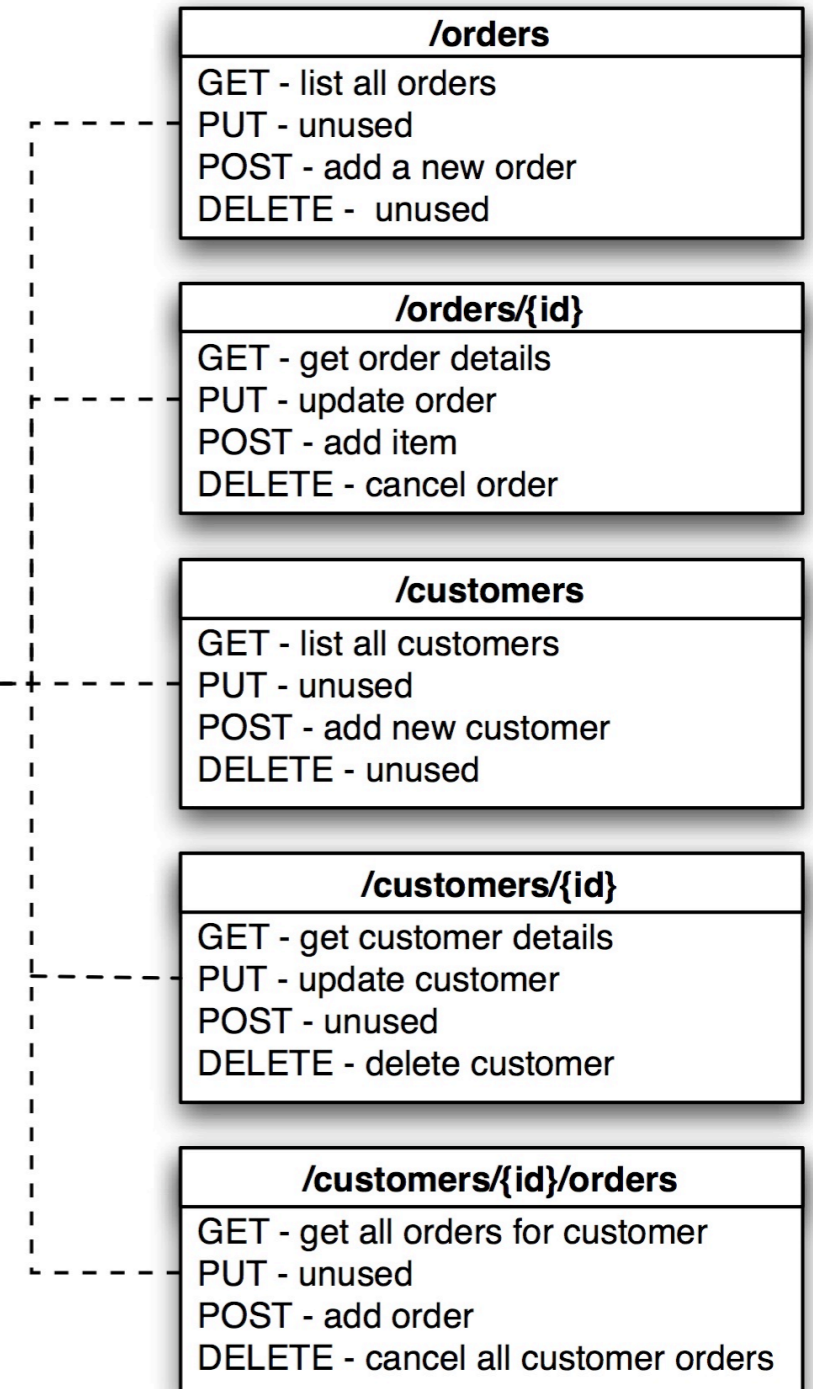
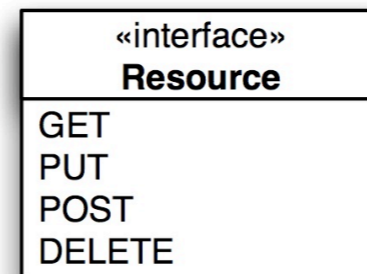
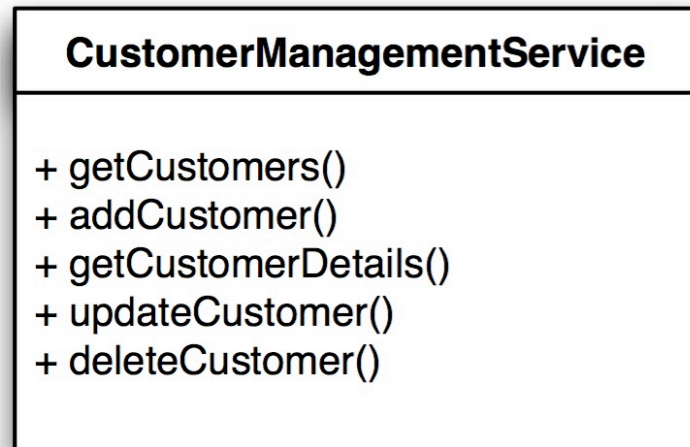
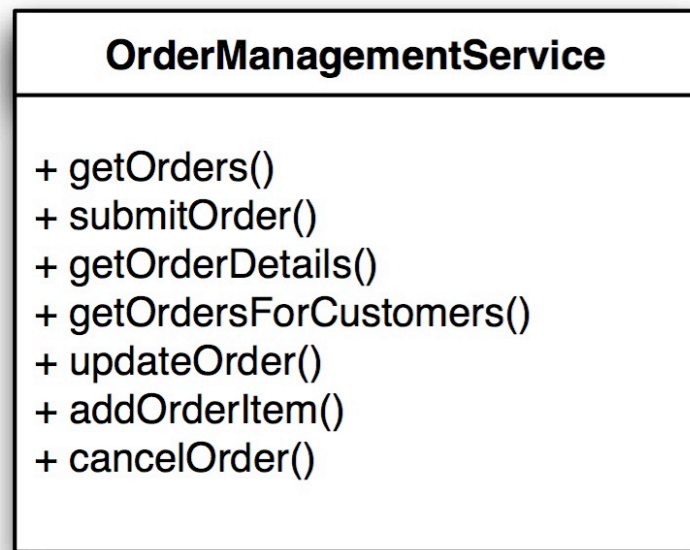
...

end:vcard

5. Communicate Statelessly



Consequences



What's cool about REST?

**A very rough analogy
(in pseudocode)**

generic

```
interface Resource {  
    Resource(URI u)  
    Response get()  
    Response post(Request r)  
    Response put(Request r)  
    Response delete()  
}
```

Any HTTP client
(Firefox, IE, curl, wget)

Any HTTP server

Caches

Proxies

Google, Yahoo!, MSN

```
class CustomerCollection : Resource {  
    ...  
    Response post(Request r) {  
        id = createCustomer(r)  
        return new Response(201, r)  
    }  
    ...  
}
```

Anything that knows
your app

specific

generic

```
interface Resource {  
    ...  
}
```

Anything that
understands HTTP

```
class AtomFeed : Resource {  
    AtomFeed get()  
    post(Entry e)  
    ...  
}
```

Any feed reader
Any AtomPub client
Yahoo! Pipes

```
class CustomerCollection : AtomFeed {  
    ...  
}
```

Anything that knows
your app

specific

REST & Rails

Rails < 2.0

```
ActionController::Routing::Routes.draw do |map|  
  # ...  
  map.connect ':controller/:action'  
end
```

http://localhost:3000/demo/read_something?value1=...&value2=...

```
class DemoController < ApplicationController  
  
  def read_something  
    # retrieve some result using params[:value1], params[:value2], ...  
  end  
  
  def change_something  
    # update backend using params[:value1], params[:value2], ...  
  end  
end
```

Rails < 2.0

Default (incl. scaffolding) unRESTful

URIs identify *actions*

No difference between POST and GET by default

Typical PHP/Java Web programming model

Rails \geq 2.0

RESTful scaffolding as default

REST considered best practice

Uniform resource interface mapped to
standard-style controllers

Routing

Simple:

```
map.resources :orders
```

Nested:

```
map.resources :customers do |customers|  
  customers.resources :orders  
end
```

Prefixed:

```
map.resources :comments, :path_prefix => '/articles/:article_id'
```

Scaffolding

```
$ script/generate scaffold Order customer_id:integer description:string
  exists app/models/
  exists app/controllers/
  exists app/helpers/
  create app/views/orders
  exists app/views/layouts/
  exists test/functional/
  exists test/unit/
  create app/views/orders/index.html.erb
  create app/views/orders/show.html.erb
  create app/views/orders/new.html.erb
  create app/views/orders/edit.html.erb
  create app/views/layouts/orders.html.erb
  identical public/stylesheets/scaffold.css
  dependency model
    exists app/models/
    exists test/unit/
    exists test/fixtures/
    create app/models/order.rb
    create test/unit/order_test.rb
    create test/fixtures/orders.yml
    exists db/migrate
    create db/migrate/002_create_orders.rb
    create app/controllers/orders_controller.rb
    create test/functional/orders_controller_test.rb
    create app/helpers/orders_helper.rb
    route map.resources :orders
```

Controller Code

```
class OrdersController < ApplicationController
  # GET /orders
  # GET /orders.xml
  def index
  end

  # GET /orders/1
  # GET /orders/1.xml
  def show
  end

  # GET /orders/new
  # GET /orders/new.xml
  def new
  end

  # GET /orders/1/edit
  def edit
  end

  # POST /orders
  # POST /orders.xml
  def create
  end

  # PUT /orders/1
  # PUT /orders/1.xml
  def update
  end

  # DELETE /orders/1
  # DELETE /orders/1.xml
  def destroy
  end
end
```

Alternatives

resource_controller:

```
class PostsController < ResourceController::Base  
end
```

http://svn.jamesgolick.com/resource_controller/

make_resourceful:

```
class FooController < ApplicationController  
  make_resourceful do  
    actions :all  
  end  
end
```

<http://mr.hamptoncatlin.com/>

Generated Routes

Output of rake routes:

orders	GET	/orders	{:controller=>"orders", :action=>"index"}
formatted_orders	GET	/orders.:format	{:controller=>"orders", :action=>"index"}
	POST	/orders	{:controller=>"orders", :action=>"create"}
	POST	/orders.:format	{:controller=>"orders", :action=>"create"}
new_order	GET	/orders/new	{:controller=>"orders", :action=>"new"}
formatted_new_order	GET	/orders/new.:format	{:controller=>"orders", :action=>"new"}
edit_order	GET	/orders/:id/edit	{:controller=>"orders", :action=>"edit"}
formatted_edit_order	GET	/orders/:id/edit.:format	{:controller=>"orders", :action=>"edit"}
order	GET	/orders/:id	{:controller=>"orders", :action=>"show"}
formatted_order	GET	/orders/:id.:format	{:controller=>"orders", :action=>"show"}
	PUT	/orders/:id	{:controller=>"orders", :action=>"update"}
	PUT	/orders/:id.:format	{:controller=>"orders", :action=>"update"}
	DELETE	/orders/:id	{:controller=>"orders", :action=>"destroy"}
	DELETE	/orders/:id.:format	{:controller=>"orders", :action=>"destroy"}

ActiveResource

Client for RESTful Rails Resources

Basic support for ActiveRecord-style code

```
class Person < ActiveResource::Base
  self.site = "http://api.people.com:3000/"
end
```

```
# Find a person with id = 1
ryan = Person.find(1)
Person.exists?(1) #=> true
```

REST ≠ CRUD

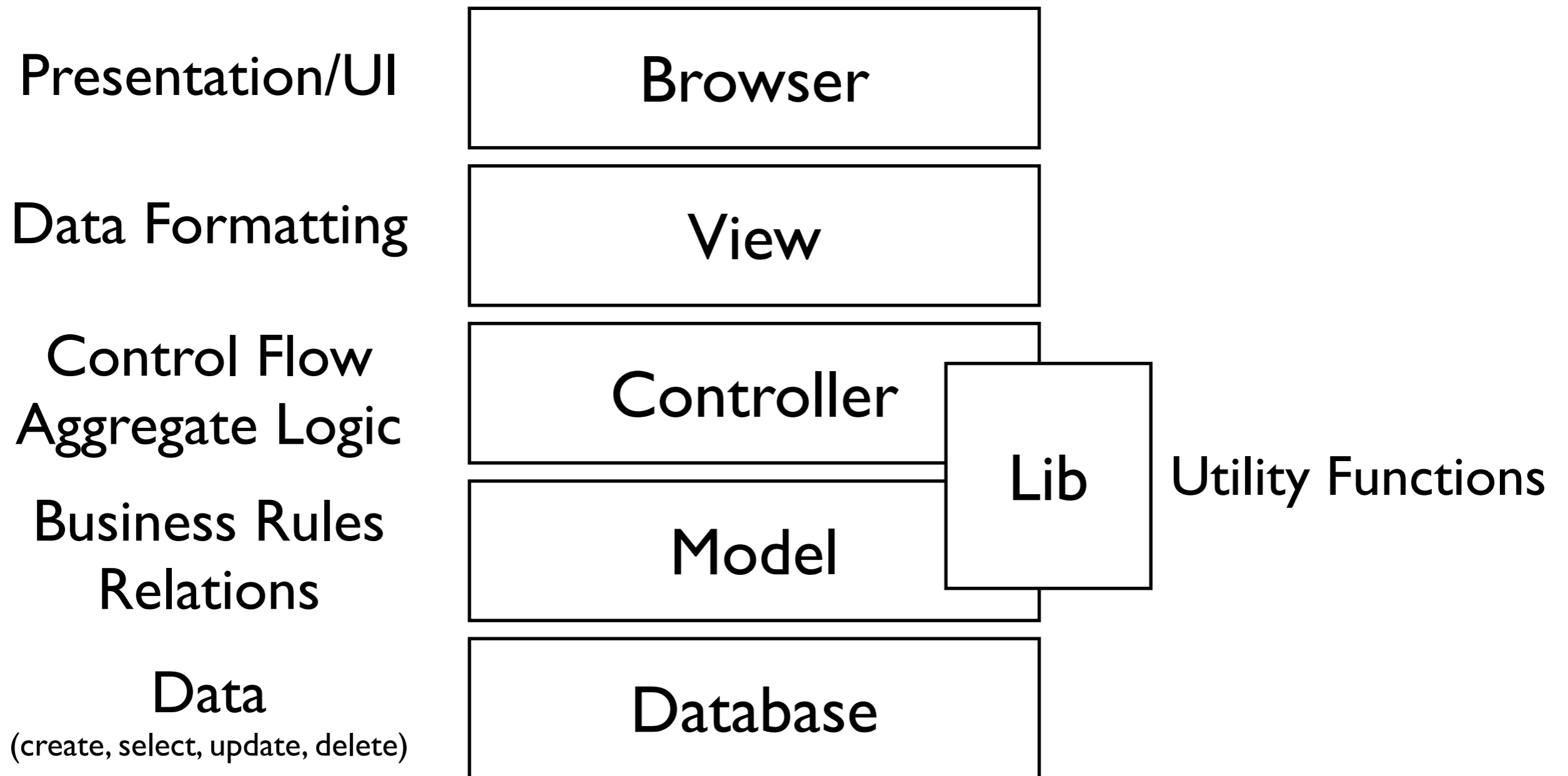
?

Resource \neq Entity

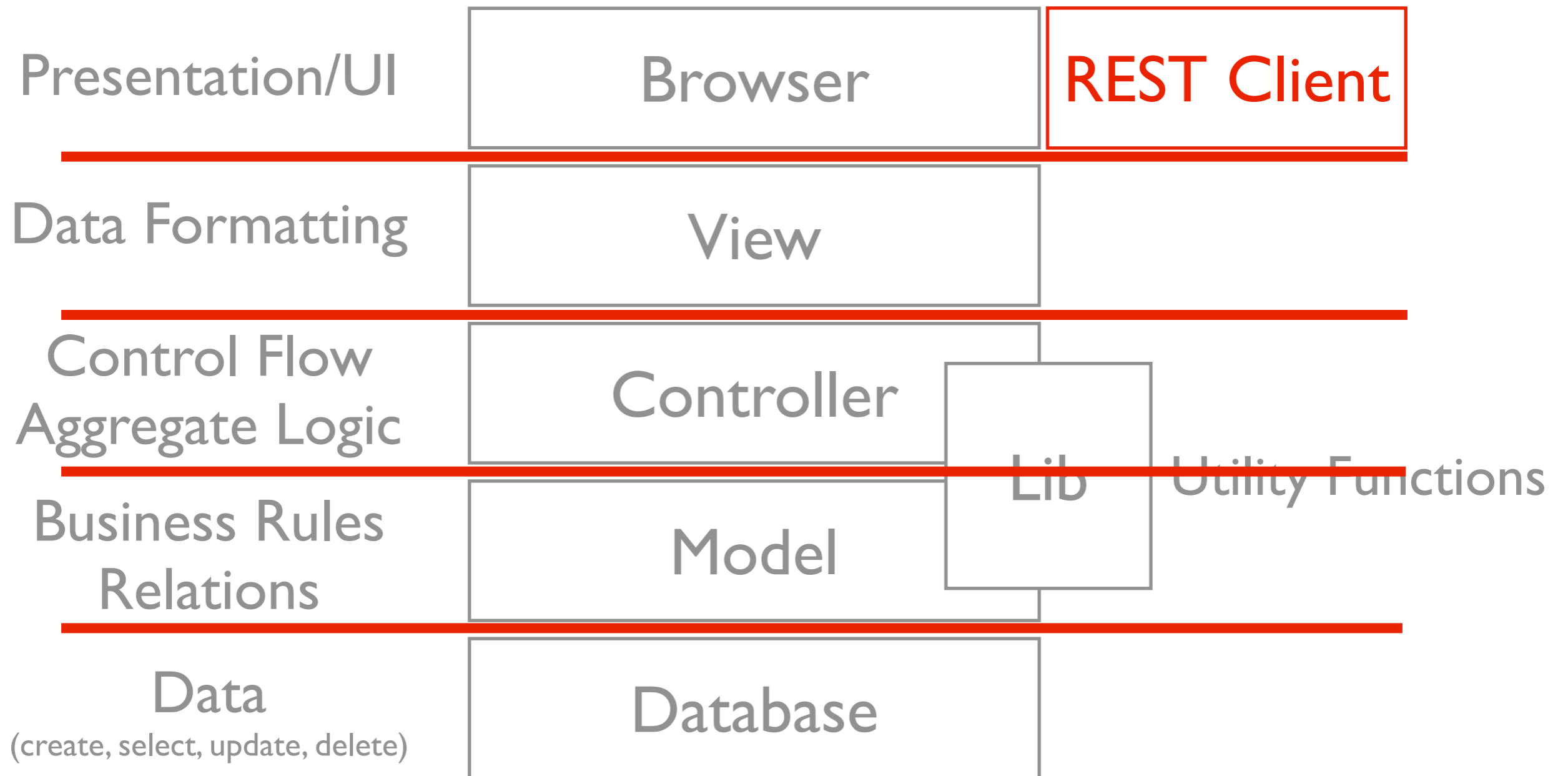
Resource \approx Model

Resource \approx Controller

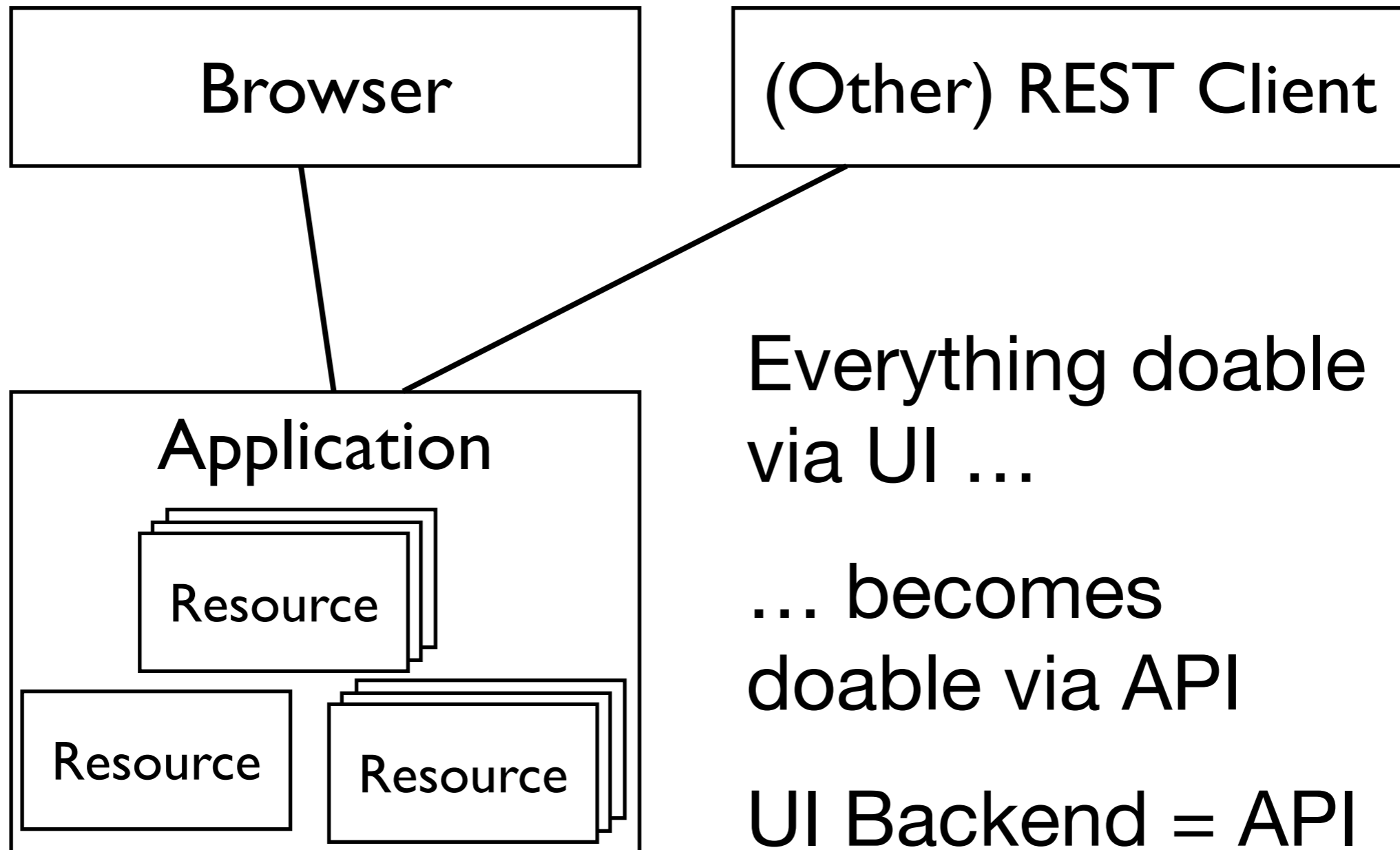
Rails App Layers



Rails App Layers & Resources



Single Resource Model



RESTful APIs

RESTful APIs don't expose low-level details

Same layer – different abstraction

Value through uniformity and hypermedia

Mapping necessity: “Implement” HTTP base interface

Mapping Examples

getFreeTimeSlots(Person)	→ GET /people/{id}/timeslots?state=free
rejectApplication(Application)	→ POST /rejections <application>http://...</application> <reason>Unsuitable for us!</reason>
performTariffCalculation(Data)	→ POST /calculations Data ← Location: http://.../calculations/47 → GET /calculations/47 ← Result
shipOrder(ID)	→ PUT /orders/08 5 <status>shipped</status>
shipOrder(ID) [variation]	→ POST /shipments Data ← Location: http://.../shipments/47

RESTful HTTP's Role for Rails

Application Size	Modularization
0-50 LOC	1 file
50-500 LOC	few files, many functions
500-1000 LOC	library, class hierarchy
1000-2000 LOC	framework + application
>2000 LOC	more than one application

How RESTful is Rails?

Positive:

Consistent and clean CRUD mapping

Use of URIs for resource identification

Support for content negotiation

Reasonable Status codes

ETags (!)

How RESTful is Rails?

Negative:

No hypermedia

No deep ETags

CRUD-centric

Proprietary protocol for ActiveRecord

My Rails/REST Wishlist

A really cool, meta-driven hypermedia programming model

for both client and server (w/o coupling)

Atom Syndication and Atom Pub Support

**If You Want to Know
More**

<http://www.innoq.com/resources/REST>



<http://www.oreilly.com/catalog/9780596529260/>

Welcome, Stefan!
Sign out
Preferences
About us
Personal feed
Home

Your Communities

- Java
- .NET
- Ruby
- SOA
- Agile
- Architecture

Featured Topics
Performance & Scalability

SOA Governance

Public Beta
Now Available

New & Notable Written for InfoQ by the Community Contribute News

DynamicJasper: Runtime generation of Jasper Reports

Community [Java](#) Topics [Open Source](#)

DynamicJasper, an open-source API which provides runtime generation of Jasper Reports, recently released version 1.3. InfoQ took the opportunity to learn more about this product, and what it provides for users. By [Ryan Slobojan](#) on Oct 08 [Discuss](#)

Presentation: Architecture Evaluation in Practice

Community [Architecture](#) Topics [Delivering Quality, Enterprise Architecture](#)

Dragos Manolescu shares insights gained from growing ThoughtWorks' architecture evaluation practice and evaluating several architectures for Global 1000 companies. These insights aim at preparing people interested in commissioning an architecture evaluation for participating in, or an evaluation to tackle the... [Marinescu](#) on Oct 08 [Discuss](#)

Ruby and the hype cycle

Community [Ruby](#) Topics [Performance & Scalability, Ruby on Rails, Stories & Case Studies](#)

A recent blog post on a failed Rails project caused a big debate about the viability of Ruby on Rails. A closer look at the post paints a different picture, though. We take a look at the reactions in the Ruby community, and compare this discussion with the upheaval about Twitter earlier this year. By [Werner Schuster](#) on Oct 08 [3 comments](#)

Adobe Max 2007 North America - Wrap Up

Community [Java](#) Topics [Rich Client / Desktop, Acquisitions, Rich Internet Apps](#)

Adobe was busy this week showing off their latest work at the 2007 Max Conference. Adobe continues to cater to developers with many of their efforts. The conference came with a number of interesting and exciting announcements for the developer community including: By [Jon Rose](#) on Oct 05 [Discuss](#)

Sponsored Links

- [The Scalability Revolution](#) SBA and the end of tier-based computing.
- [Rule your SOA](#)

- [Deploy Ajax & Flash Apps to the Desktop.](#) Free download of Adobe Integrated Runtime Beta.

Exclusive Content

Steve Sloan on BizTalk Server 2006 R2

InfoQ talked to Steve Sloan, Senior Product Manager, about the BizTalk Server 2006 R2 in the context of SOA. [SOA](#), Oct 04, 2007, [Discuss](#)

Open Source WS Stacks for Java - Design Goals and Philosophy

InfoQ spoke to the lead developers of the most important open source Java Web-services stacks about their design goals, standards, data binding, XML, interoperability, REST support, and maturity. [SOA, Java](#), Oct 04, 2007, [7](#)

Creating dynamic web applications with JSF/DWR/DOJO

JSF, DWR, and Dojo are all popular technologies in their own right. This article looks at how they can be integrated together in a portal environment. [Java](#), Oct 04, 2007, [1](#)

Architecture Evaluation in

http://www.infoq.com

All
Articles
Presentations
Interviews
Books

Recommendations

Learn more about REST

Learn to love the URI

Adopt REST principles for UIs ...

...APIs will follow on their own

Appreciate the Web

Thank you!
Any questions?

<http://www.innoq.com>
<http://railsconsulting.de>

Stefan Tilkov

<http://www.innoq.com/blog/st/>



Architectural Consulting

SOA	WS-*	REST
MDA	MDSD	MDE
J(2)EE	RoR	.NET

innoQ Deutschland GmbH
Halskestraße 17
D-40880 Ratingen
Phone +49 21 02 77 162-100
info@innoq.com · www.innoq.com

innoQ Schweiz GmbH
Gewerbestrasse 11
CH-6330 Cham
Phone +41 41 743 01 11