A Pragmatic Introduction to REST

Stefan Tilkov, stefan.tilkov@innoq.com

Stefan Tilkov

http://www.innoQ.com

stefan.tilkov@innoq.com

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

http://www.InfoQ.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

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

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

```
<customer><order ref='./orders/46'</customer>
```

shutdown

update software

replace hardware

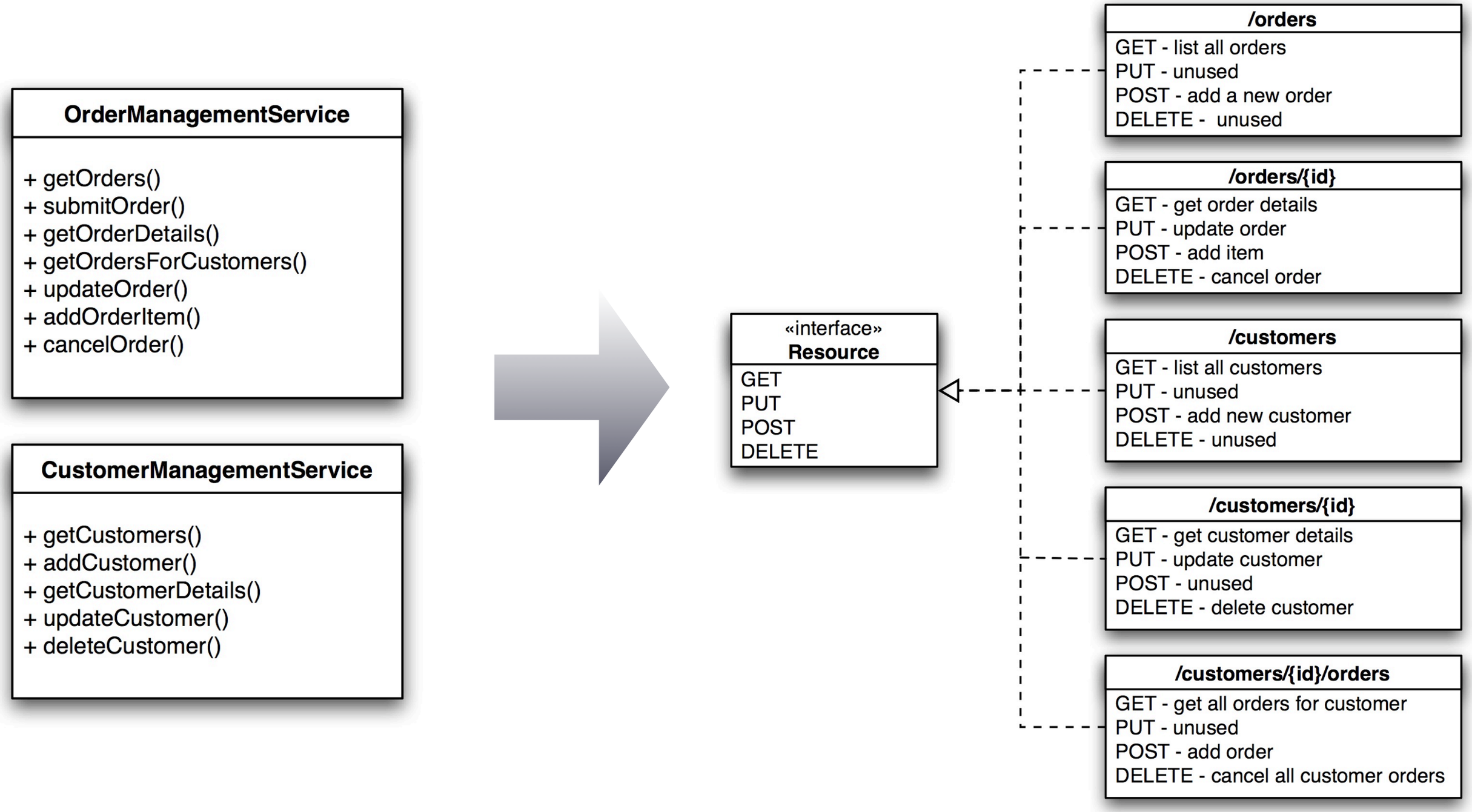
startup

```
GET /customers/1234/orders/46
Host: example.com
Accept: application/vnd.mycompany.order+xml
```

```
<order>...</order>
```

time

Consequences

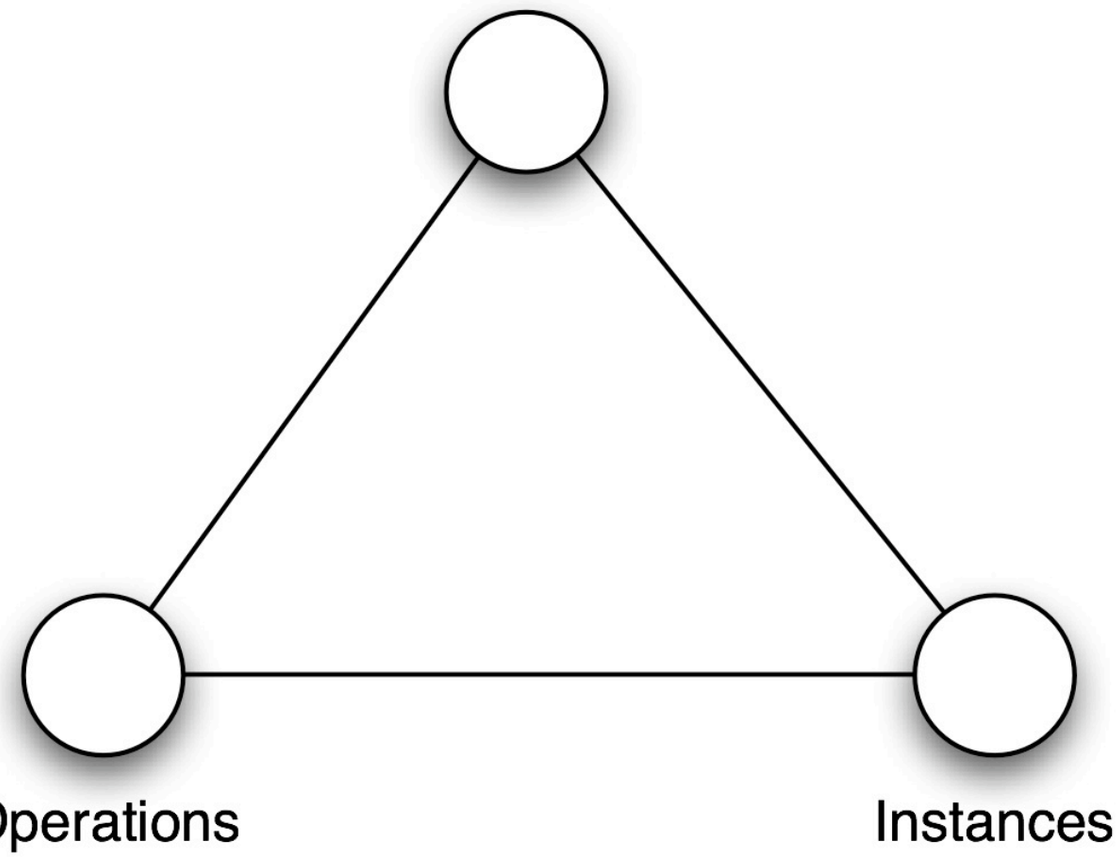

OrderManagementService
+ getOrders()
+ submitOrder()
+ getOrderDetails()
+ getOrdersForCustomers()
+ updateOrder()
+ addOrderItem()
+ cancelOrder()
CustomerManagementService
+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()
«interface»
Resource
GET
PUT
POST
DELETE
/orders
GET - list all orders
PUT - unused
POST - add a new order
DELETE - unused
/orders/{id}
GET - get order details
PUT - update order
POST - add item
DELETE - cancel order
/customers
GET - list all customers
PUT - unused
POST - add new customer
DELETE - unused
/customers/{id}
GET - get customer details
PUT - update customer
POST - unused
DELETE - delete customer
/customers/{id}/orders
GET - get all orders for customer
PUT - unused
POST - add order
DELETE - cancel all customer orders

Cheating?

Maybe.

many

Data types



many

very few

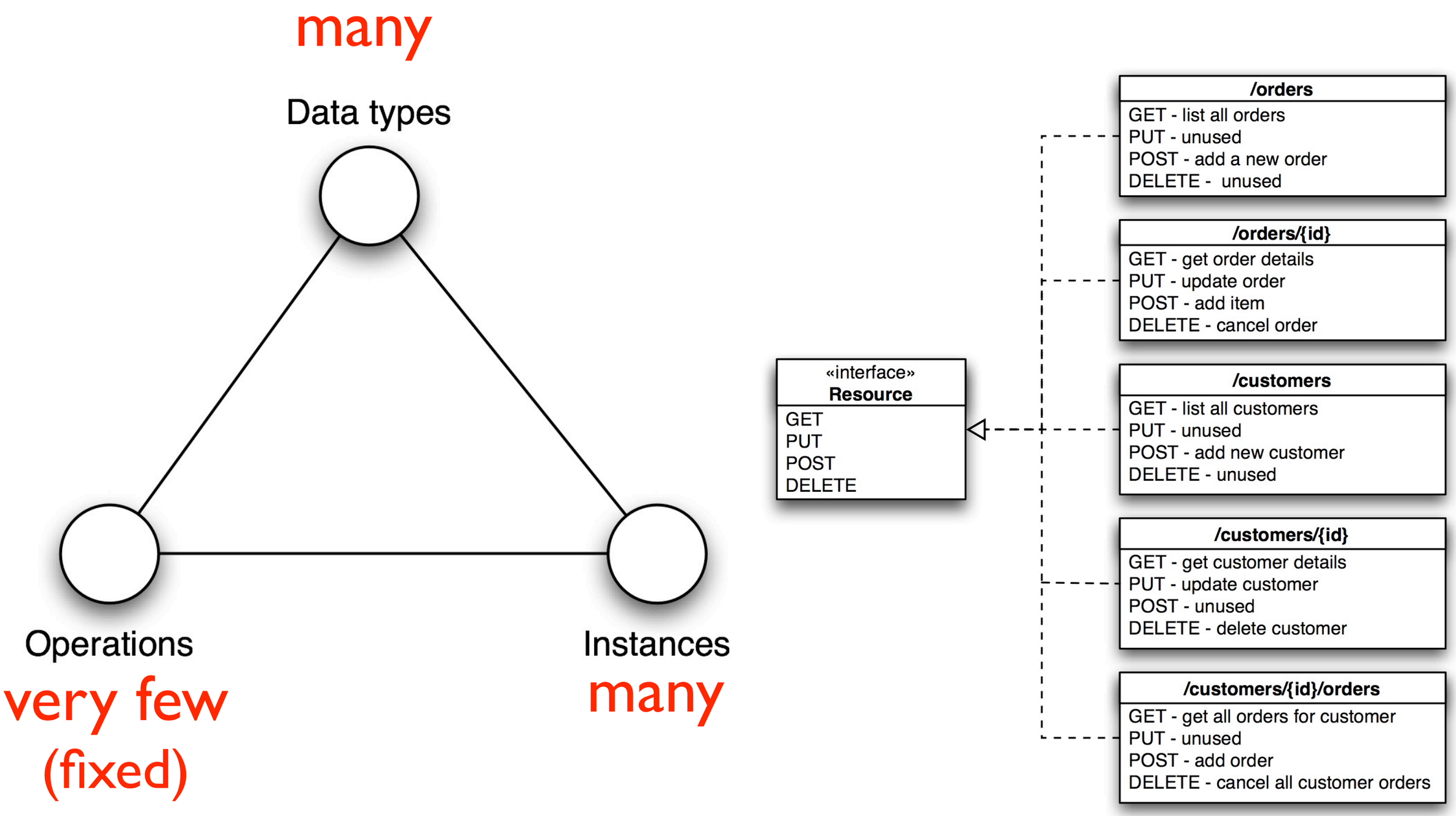
(one per service)

OrderManagementService

+ getOrders()
+ submitOrder()
+ getOrderDetails()
+ getOrdersForCustomers()
+ updateOrder()
+ addOrderItem()
+ cancelOrder()

CustomerManagementService

+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()


many
Data types
Operations
very few
(fixed)
Instances
many
«interface»
Resource
GET
PUT
POST
DELETE
/orders
GET - list all orders
PUT - unused
POST - add a new order
DELETE - unused
/orders/{id}
GET - get order details
PUT - update order
POST - add item
DELETE - cancel order
/customers
GET - list all customers
PUT - unused
POST - add new customer
DELETE - unused
/customers/{id}
GET - get customer details
PUT - update customer
POST - unused
DELETE - delete customer
/customers/{id}/orders
GET - get all orders for customer
PUT - unused
POST - add order
DELETE - cancel all customer orders

Designing a RESTful Application

Identify resources & design URIs

Select formats (or create new ones)

Identify method semantics

Select response codes

See: http://bitworking.org/news/How_to_create_a_REST_Protocol

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

Some HTTP Features

Verbs (in order of popularity):

- GET, POST
- PUT, DELETE
- HEAD, OPTIONS, TRACE

Standardized (& meaningful) response codes

Content negotiation

Redirection

Caching (incl. validation/expiry)

Compression

Chunking

RESTful HTTP Advantages

Universal support (programming languages, operating systems, servers, ...)

Proven scalability

Real web integration for machine-2-machine communication

Support for XML, but also other formats

REST and Web Services

(very briefly, I promise)

Web Services Issues

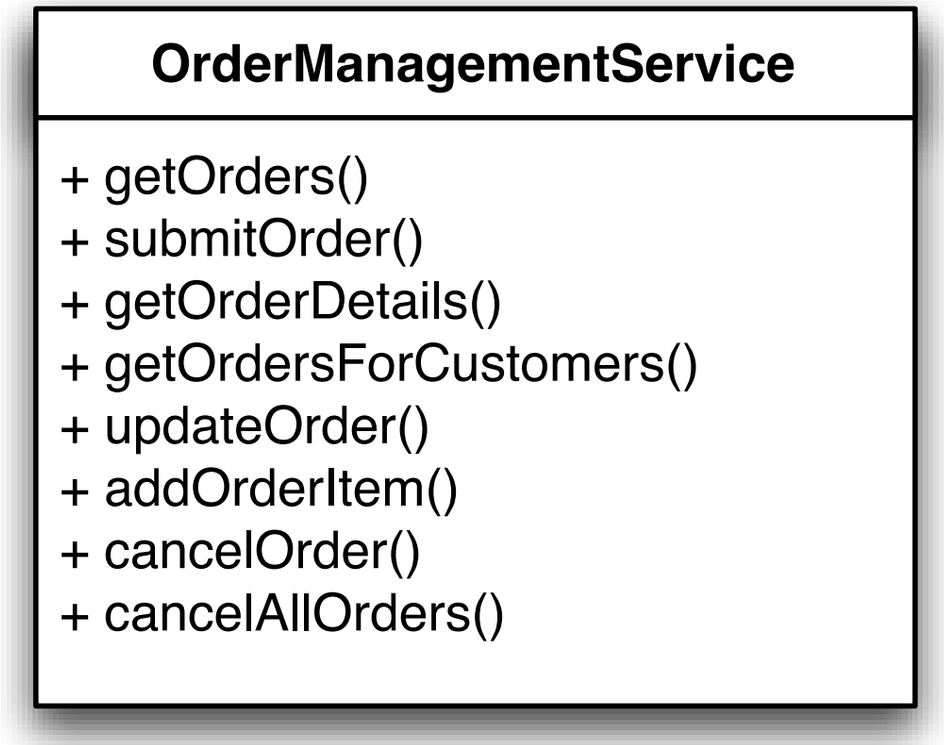
Web Services are "Web" in name only

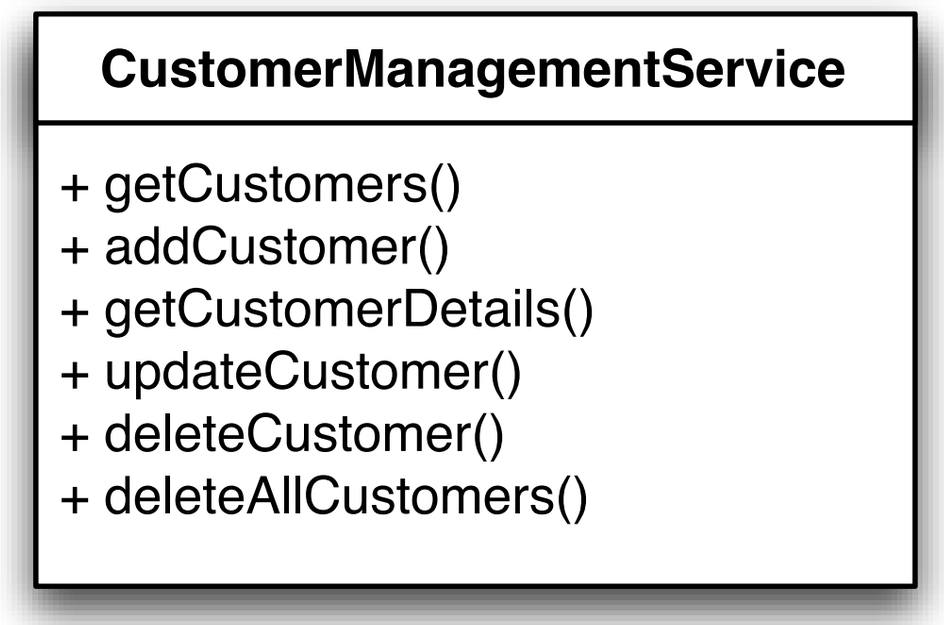
WS-* tends to ignore the web

Abstractions leak, anyway

Protocol independence is a bug, not a feature

Web Services





A separate interface (façade) for each purpose

As known CORBA, DCOM, RMI/EJB

Often used for SOA (“CORBA w/ angle brackets)

Application-specific protocol

Contribution to the Net's Value

2 URLs

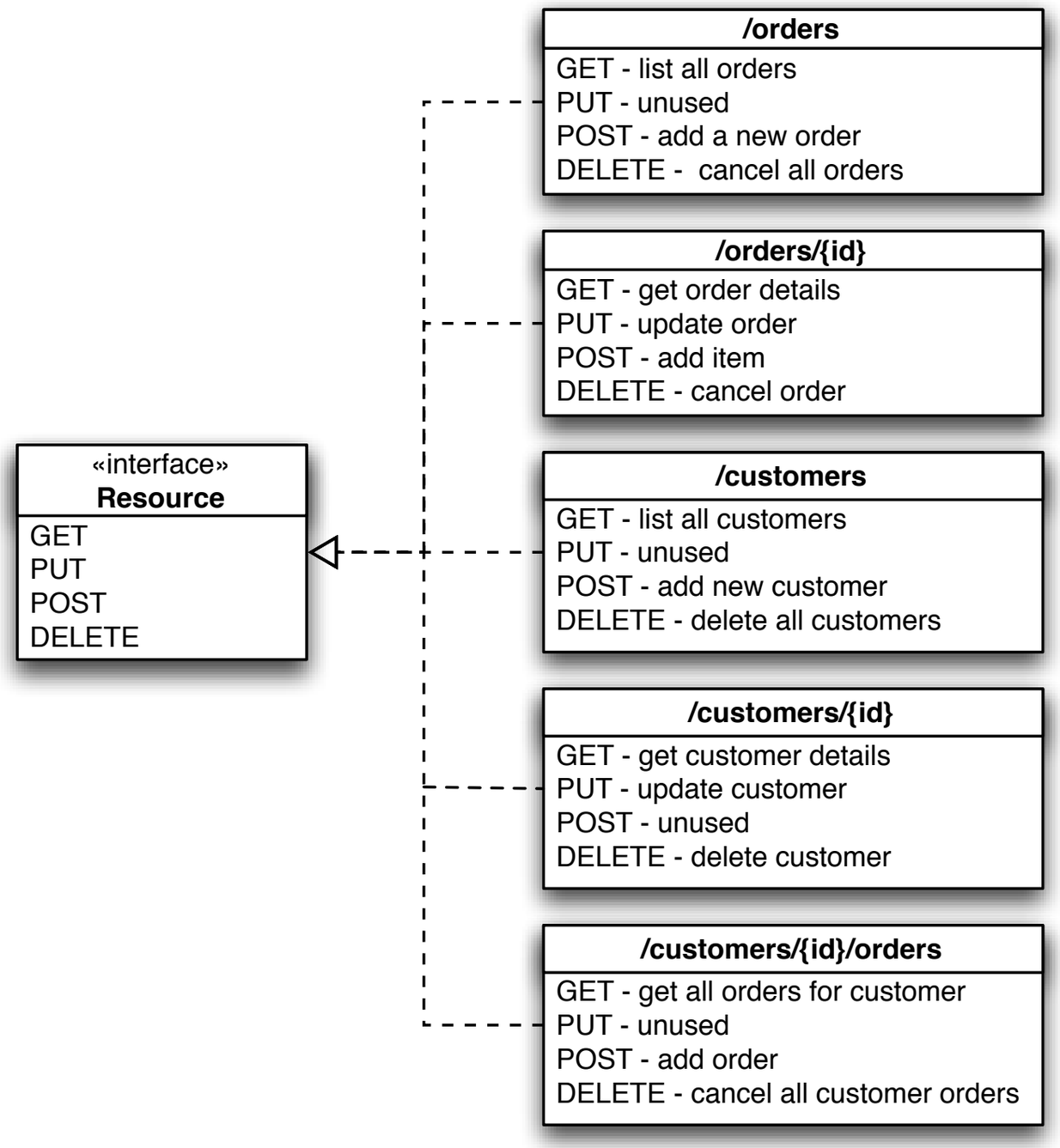
http://example.com/customerservice

http://example.com/orderservice

1 method

POST

REST Approach



A single *generic* (uniform) interface for everything

Generic verbs mapped to resource semantics

A standard application protocol (e.g. HTTP)

Contribution to the Net's Value

Millions of URLs

- every customer
- every order

4-6 supported methods per resource

- GET, PUT, POST, DELETE, OPTIONS, HEAD

Cacheable, addressable, linkable, ...

REST for SOA

Business	SOA as an approach to business/IT alignment	
Architecture	Technical SOA	REST
Technology	SOAP, WSDL, WS-*	(RESTful) HTTP, URI, ...

REST as an alternative way to achieve SOA goals

REST & Rails

Rails < 2.0

```
ActionController::Routing::Routes.draw do |map|
  # ...
  map.connect ':controller/:action/:id'
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 ≥ 2.0

```ruby
ActionController::Routing::Routes.draw do |map|
  map.resources :orders
end
```

```
          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"}
```

Demo

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 ActiveResource

My Rails/REST Wishlist

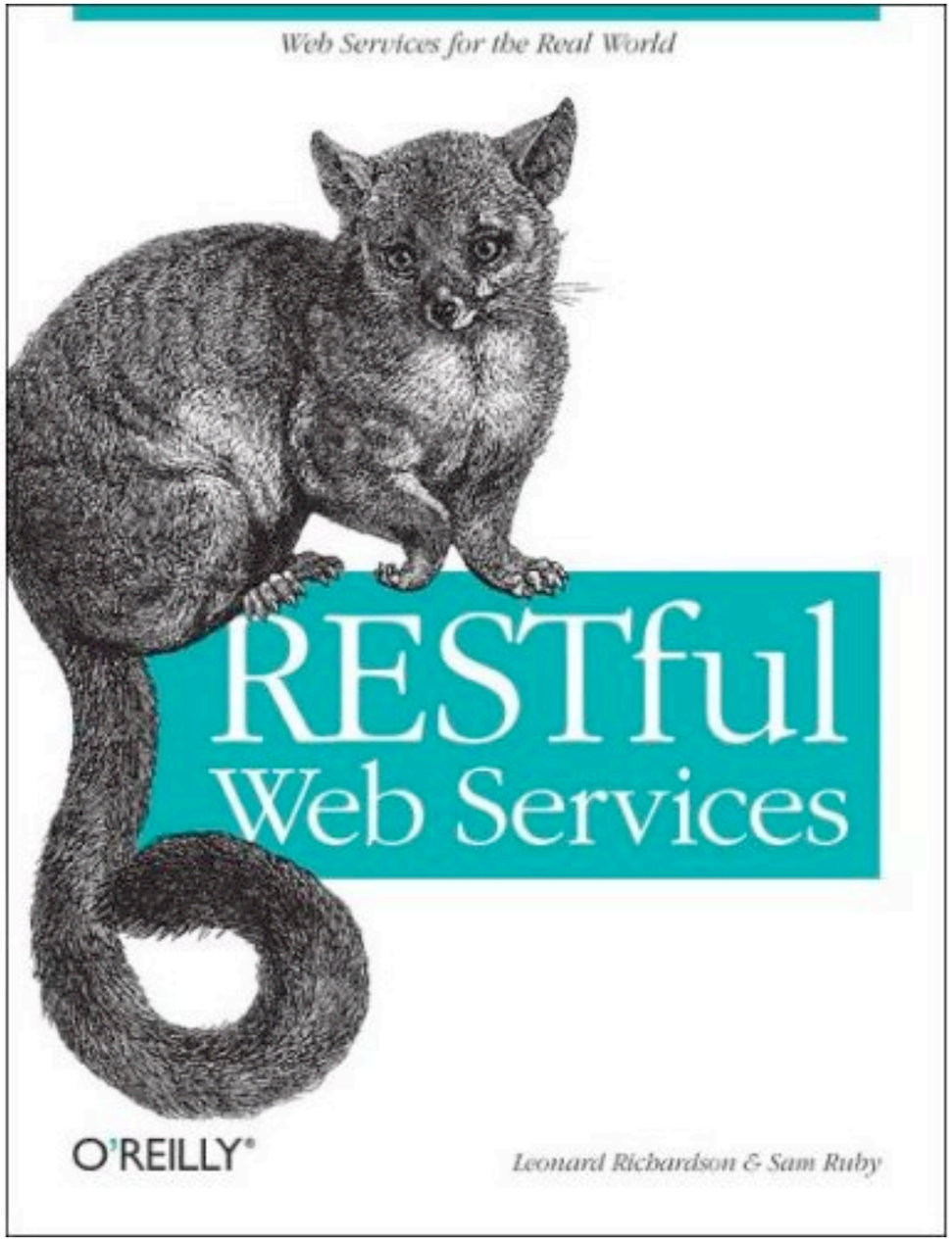
A really cool, meta-driven hypermedia programming model

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/

http://www.infoq.com/

Random Stuff | innoQ | InfoQ | SOA-Expertenwissen | Txp › SOA-E...ertenwissen | innoQ | Partner | banking | awstats | js | 1&1 | O1BC.DE | AR | RailsDocs

InfoQ

Tracking change and innovation in the enterprise software development community

Version 1.4

En | 中文 | 日本語

174,255 Sep unique visitors

Welcome, Stefan!

Sign out
Preferences
About us
Personal feed
Home

Your Communities

- [x] Java
- [ ] .NET
- [x] Ruby
- [x] SOA
- [ ] Agile
- [x] Architecture

Search

Featured Topics

Performance & Scalability

SOA Governance

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 commiss
participating in, or an
evaluation to tackle th
Marinescu on Oct 08 D

http://www.infoq.com

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

Exclusive Content

All | Articles | Presentations | Interviews | Books

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,

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

Thank you!

Stefan Tilkov
http://www.innoq.com/blog/st/
stefan.tilkov@innoq.com

innoQ

innoQ Deutschland GmbH
Halskestraße 17
D-40880 Ratingen
Phone +49 21 02 77 162-100

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

info@innoq.com · www.innoq.com

Photo Credit

http://en.wikipedia.org/wiki/Image:Sangreal.jpg