> *What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is there some broken manual somewhere that needs to be fixed?*
>
> *— Roy T. Fielding*

REST Assured

Hypermedia APIs with Spring

Oliver Gierke

Oliver Gierke

Engineer @ Pivotal

Spring Data project lead

ogierke@gopivotal.com

www.olivergierke.de

olivergierke

Background

REST

REST

Resources

URIs

Uniform Interface

Representations

Hypermedia

"HATEOAS - the word, there's no pronunciation for.

– Ben Hale (SpringOne2GX 2012)

Hypermedia

Links in representations

State navigations discoverable

```
{ _links : {
    self : { href : … },
    cancel : { href : … },
    update : { href : … },
    payment : { href : "…/orders/4711/payment" }
  },

  items : [ {
        …
  } ],

  location : "take-away",
  price : 4.2,
  status : "payment expected"
}
```

MediaType

text/html

application/hal+(json|xml)

application/collection+json

Implementation aspects

Hypermedia VS. Java Frameworks

	Spring MVC	JAX-RS	
HTTP Methods	✓	✓	
URI Mapping	✓	✓	
Content negotiation	✓	✓	
Hypermedia	?	?	

Spring HATEOAS

Spring HATEOAS

Representation models

`LinkBuilder` API

Representation enrichment

http://bit.ly/spring-hateoas

DEMO

https://github.com/olivergierke/spring-hateoas-sample

Spring Data REST

Spring Data REST

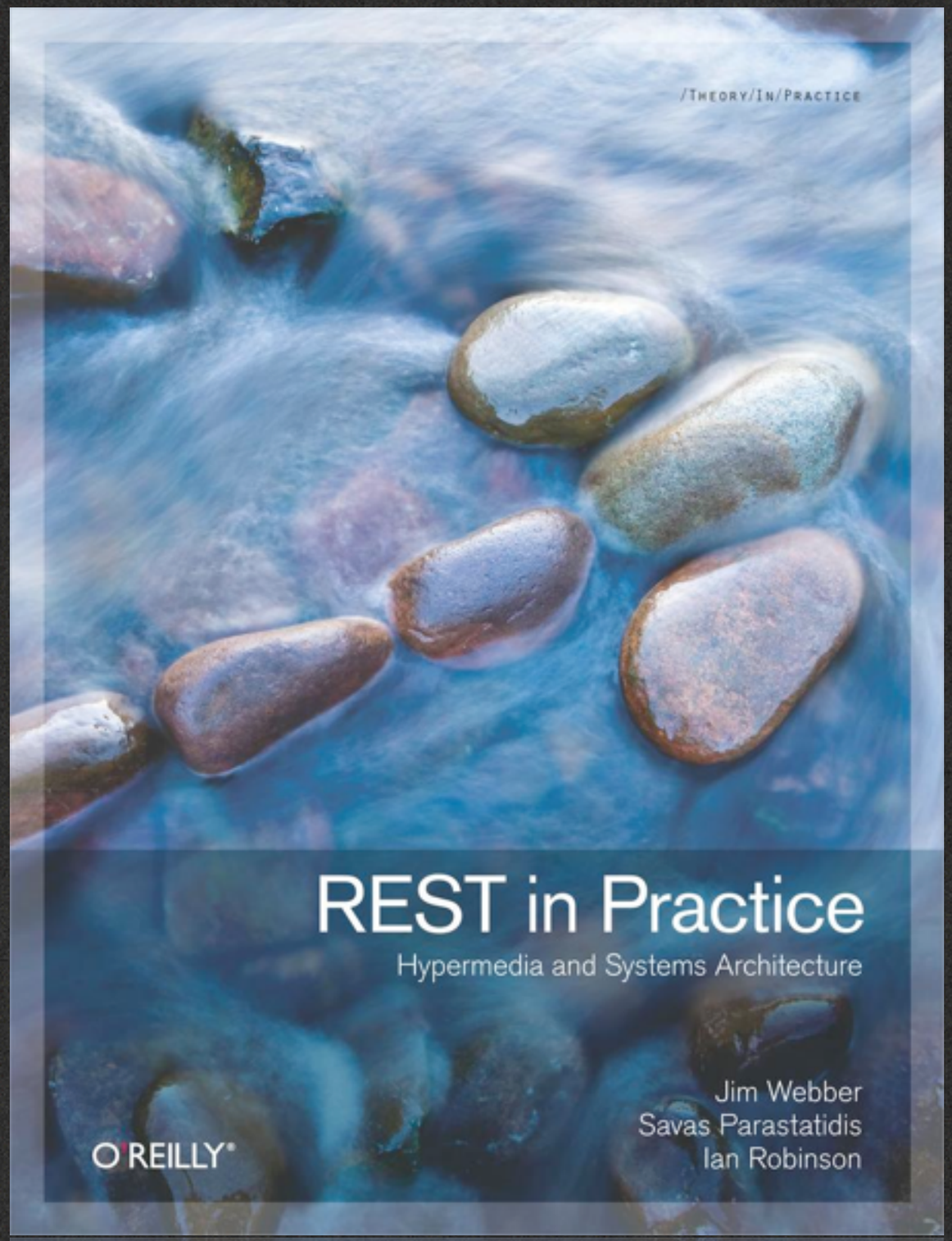
Export Spring Data repositories in a hypermedia-driven way

Do „the right thing™“ by default

DEMO

https://github.com/olivergierke/rest-microservices

REST in practice

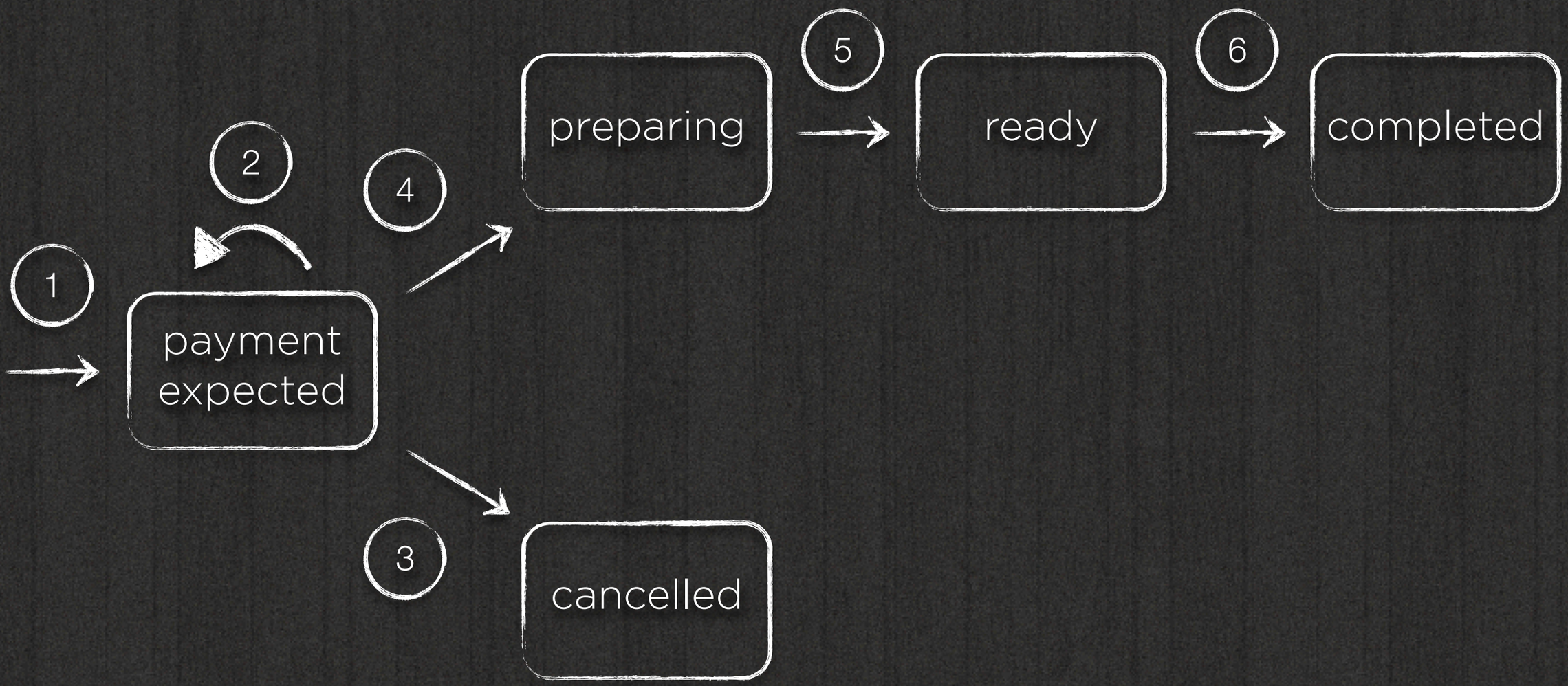

/Theory/In/Practice
REST in Practice
Hypermedia and Systems Architecture
Jim Webber
Savas Parastatidis
Ian Robinson
O'REILLY®

RESTBucks

RESTBucks

Starbucks (like) coffee ordering

Order / Payment


1
payment expected
2
3
cancelled
4
preparing
5
ready
6
completed

Method	URI	Action	Step
POST	/orders	Create new order	1
POST/PATCH	/orders/4711	Update the order (only if "payment expected")	2
DELETE	/orders/4711	Cancel order (only if "payment expected")	3
PUT	/orders/4711/payment	Pay order (only if "payment expected")	4
Barista preparing the order			
GET	/orders/4711	Poll order state	5
GET	/orders/4711/receipt	Access receipt	
DELETE	/orders/4711/receipt	Conclude the order process	6

Challenges

Challenges

How to avoid hard coding URIs?

Use link relations

orders	Returns all orders available in the system
order	Returns a single order
self	The uri value can be used to `GET` the latest resource representation of the order.
cancel	This is the URI to be used to `DELETE` the order resource should the consumer wish to cancel the order.
update	Consumers can change the order using a `POST` to transfer a representation to the linked resource.
payment	The linked resource allows the consumer to begin paying for an order. Initiating payment involves `PUT`ting an appropriate resource representation to the specified URI
receipt	The URI to access the receipt using `GET` and conclude the order by taking the receipt (use `DELETE`).

orders	Returns all orders available in the system
order	Returns a single order
self	The uri value can be used to `GET` the latest resource representation of the order.
cancel	This is the URI to be used to `DELETE` the order resource should the consumer wish to cancel the order.
update	Consumers can change the order using a `POST` to transfer a representation to the linked resource.
payment	The linked resource allows the consumer to begin paying for an order. Initiating payment involves `PUT`ting an appropriate resource representation to the specified URI
receipt	The URI to access the receipt using `GET` and conclude the order by taking the receipt (use `DELETE`).

Method	URI	Action	Step
`POST`	/orders	Create new order	1
`POST/PATCH`	/orders/4711	Update the order (only if "payment expected")	2
`DELETE`	/orders/4711	Cancel order (only if "payment expected")	3
`PUT`	/orders/4711/payment	Pay order (only if "payment expected")	4
Barista preparing the order			
`GET`	/orders/4711	Poll order state	5
`GET`	/orders/4711/receipt	Access receipt	
`DELETE`	/orders/4711/receipt	Conclude the order process	6

Method	Relation type	Action	Step
`POST`	orders	Create new order	1
`POST/PATCH`	update	Update the order (only if "payment expected")	2
`DELETE`	cancel	Cancel order (only if "payment expected")	3
`PUT`	payment	Pay order (only if "payment expected")	4
Barista preparing the order			
`GET`	order	Poll order state	5
`GET`	receipt	Access receipt	
`DELETE`	receipt	Conclude the order process	6

Challenges

How to implement:
"only if payment expected“?

Clients react on the presence of links

Spring RESTBucks

Spring RESTBucks

Sample implementation

Using Spring technologies

http://bit.ly/spring-restbucks

	Orders	Payment
Web	Spring Data REST	Manual implementation
Service	-	Manual implementation
Repository	Spring Data	Spring Data

DEMO

https://github.com/olivergierke/spring-restbucks

API docs?

Services for a Changing World

RESTful Web APIs

Leonard Richardson,
Mike Amundsen & Sam Ruby

O'REILLY®

"How does the client make sense of all this?

Profiles

Profiles

RFC 6906

Points to resources describing additional semantics within a media type

Curies

(HAL) Curies

```
{ "_links" : {

    "self" : { "href" : … },
    "restbucks:cancel" : { "href" : … },
    "restbucks:update" : { "href" : … },

    "curies" : {
      "name" : "restbucks",
      "href" : "…/rels/{rel}"
      "templated" : true
    }
  },
  …
}
```

Docs for **`update`** are at: **`…/rels/update`**

alps.io

alps.io

Application Level Profile Semantics

Describe state transitions and payloads

Media type agnostic

Repository with pre-defined docs

`application/alps+(json|xml)`

DEMO

Miscellaneous

Spring MVC integration testing

REST Shell

Thank you!

Resources

Code

Spring HATEOAS Sample

REST micro-services

Spring RESTBucks

Books

RESTful Web APIs

REST in Practice

REST und HTTP

Videos

Hypermedia APIs - Jon Moore

Hypermedia APIs with Spring