REST: Slightly more than an Introduction

Stefan Tilkov, @stilkov

JUG Ostfalen, AutoUni 2013-03-04

Wir lösen das – persönlich!

innoQ

Stefan Tilkov

stefan.tilkov@innoq.com

@stilkov

http://heise.de/developer/podcast/

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

http://www.innoq.com

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116

info@innoq.com

http://rest-http.info

innoQ

REpresentational State Transfer

Roy Fielding, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

The REST Uniform Interface

identification of resources

resource manipulation through representations

hypermedia as the engine of application state

self-descriptive messages

The REST Uniform Interface

identification of resources

resource manipulation through representations

hypermedia as the engine of application state

self-descriptive messages

```
http://example.com/orders?year=2008

http://example.com/customers/1234

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

http://example.com/products/4554

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

The REST Uniform Interface

identification of resources

resource manipulation through representations

hypermedia as the engine of application state

self-descriptive messages

```
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
```

The REST Uniform Interface

identification of resources

resource manipulation through representations

hypermedia as the engine of application state

self-descriptive messages

```
<order self='http://example.com/orders/3321'>
  <item>
    <amount>23</amount>
    <product ref='http://example.com/products/4554' />
  </item>
  <customer ref='http://example.com/customers/1234' />
  <link rel='items'
        ref='http://example.com/orders/3321/items' />
</order>
```

The REST Uniform Interface

identification of resources

resource manipulation through representations

hypermedia as the engine of application state

self-descriptive messages

Standard Method

```
GET /service/customers/1234 HTTP 1.1
Host: www.example.com
User-Agent:     XYZ 1.1
Accept:         text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Keep-Alive:     300
Connection:     keep-alive
If-Modified-Since:      Fri, 02 Oct 2009 16:47:31 GMT
If-None-Match: "600028c-59fb-474f6852c9dab"
Cache-Control: max-age=60

HTTP/1.1 200 OK
Date:  Sun, 04 Oct 2009 19:16:25 GMT
Server:  Apache/2.2.11 (Debian)
Last-Modified: Fri, 02 Oct 2009 16:47:31 GMT
Etag:  "600028c-59fb-474f6852c9dab"
Cache-Control:  max-age=300
Accept-Ranges:          bytes
Vary:   Accept-Encoding
Content-Encoding:   gzip
Content-Length:         7160
Keep-Alive:     timeout=15, max=91
Connection:   Keep-Alive
Content-Type:           application/xml

<?xml version='1.0' encoding='utf-8' ?>
...
```

Media Type

Visibility

Control Data

Data

innoQ

getOrderDetails()

submitApplicationData()

updateQuote()

findMatchingBid()

initiateProcess()

cancelSubscription()

listAuctions()

getUsers()

getOrderDetails()

findMatchingBid()

GET

listAuctions()

getUsers()

initiateProcess()

submitApplicationData()

POST

PUT

updateQuote()

DELETE

cancelSubscription()

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

generic

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

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

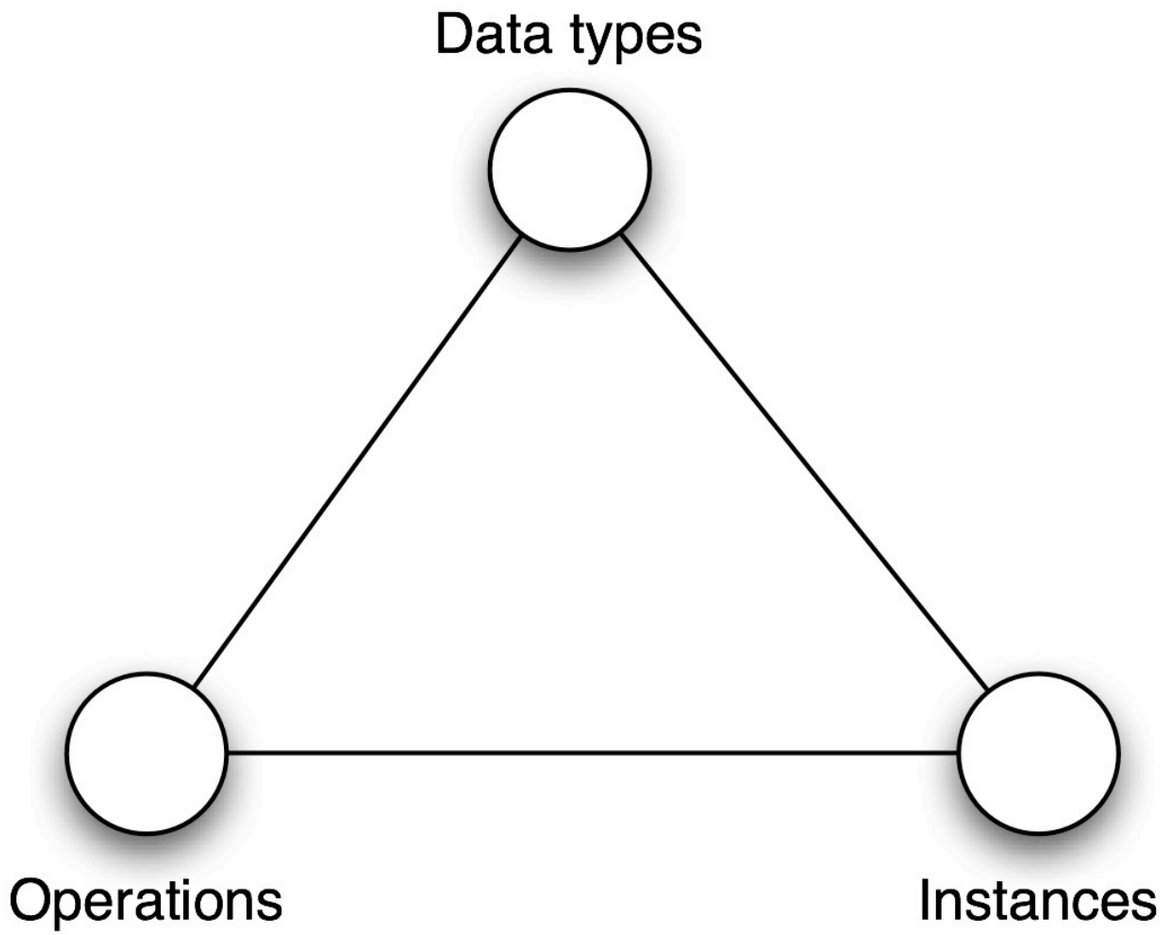
GET - get all orders for customer
PUT - unused
POST - add order
DELETE - cancel all customer orders

Common Misconceptions

Standard methods are not Enough

innoQ

many



many

very few

(one per service)

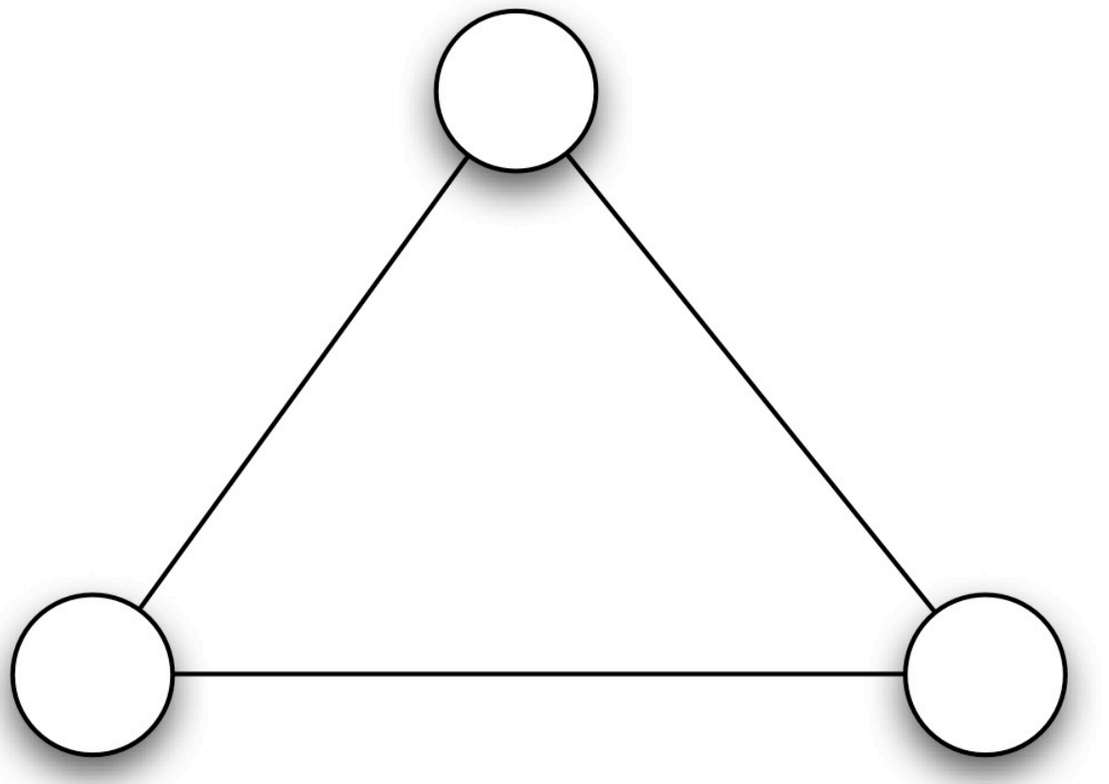
OrderManagementService

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

CustomerManagementService

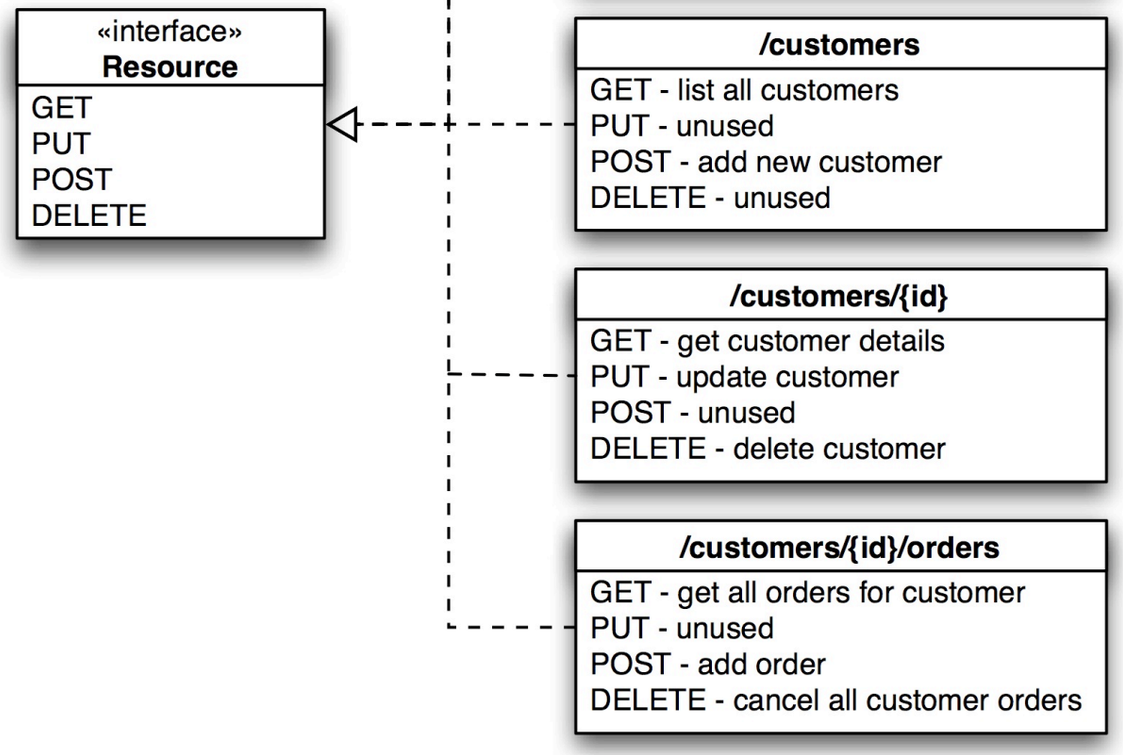
+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()

many



very few
(fixed)

many



getFreeTimeSlots(Person)	→GET /people/st/timeslots?state=free
rejectApplication(Application)	→POST /rejections↵ <application>http://...</application>↵ <reason>Unsuitable for us!</reason>
performTariffCalculation(Data)	→POST /contracts↵ Data ←Location: http://.../contracts/4711 <tariff ref='./contracts/4711/tariff' /> →GET /contracts/4711/tariff ←Result
shipOrder(ID)	→PUT /orders/0815/status↵ <status>shipped</status>
shipOrder(ID) [variation]	→POST /shipments↵ Data ←Location: http://.../shipments/4711

REST offers less standardization than SOAP/WSDL/WS-* Web services

innoQ

(“REST” is not a standard)

Standardized with RESTful HTTP

Identity

Methods

Content (Format) Negotiation

Caching

Compression

Chunking

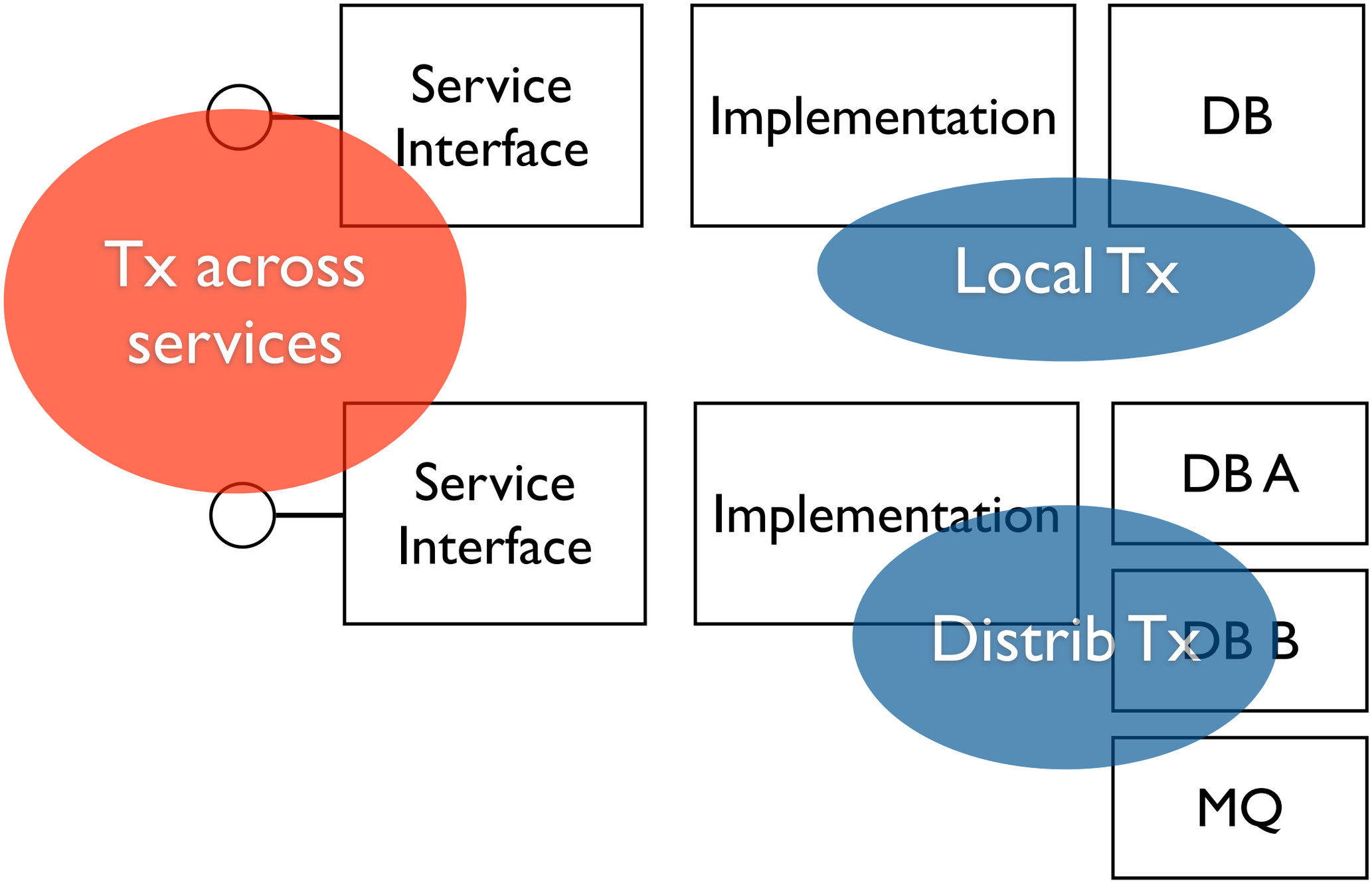
Authentication

REST is for easy things,
WS-* for complex ones

easy hard

simple complex

Transactions



create new Tx resource

```
→POST /customer-registration
←Location: http://.../
registration-4711
<status>in progress</status>
```

augment state

```
→POST /registration-4711 ↵
    ...data...
```

get state

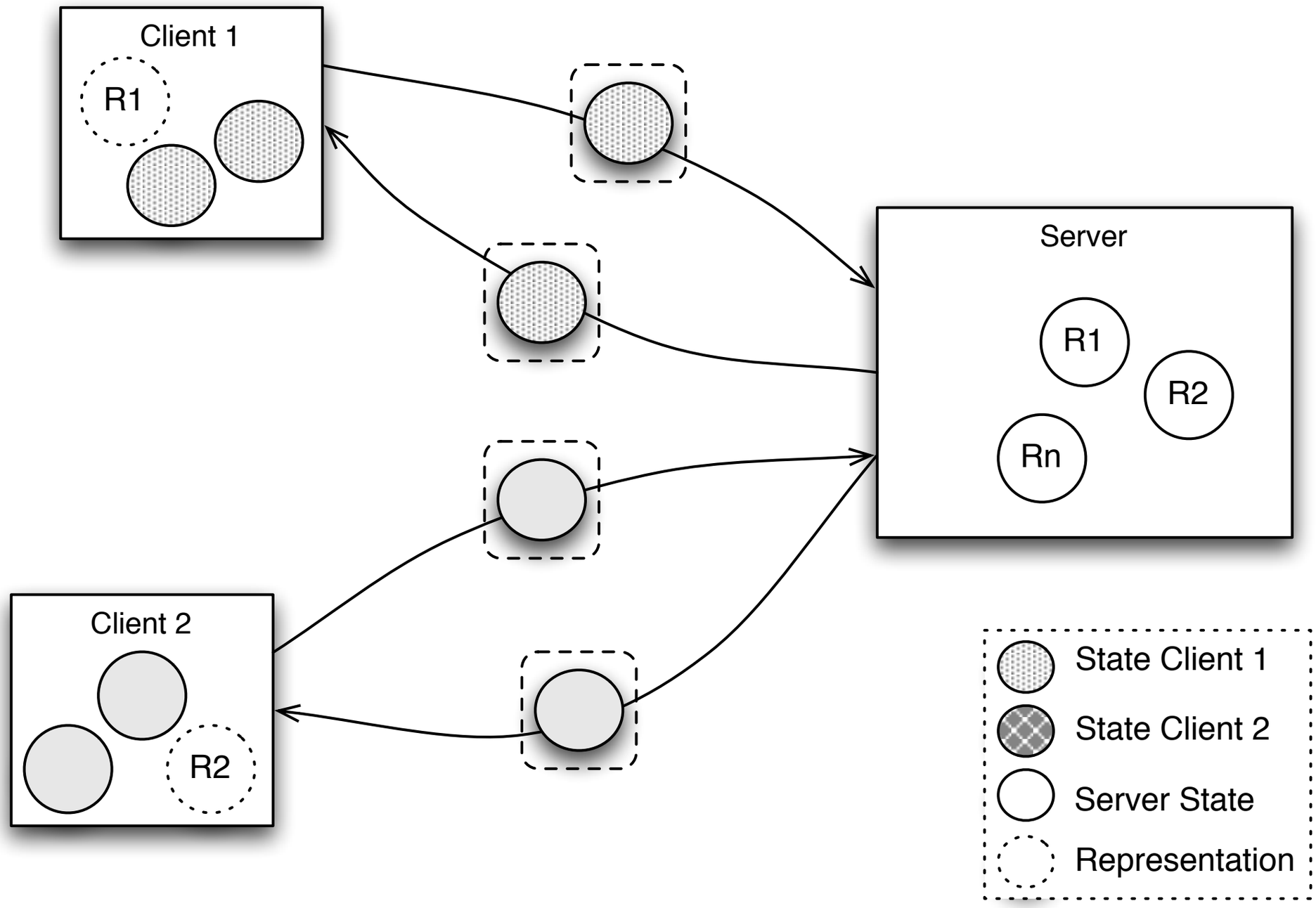
```
→GET /registration-4711
←... data ...
<link rel='status'
      href='r4711/status'>
  in progress
</link>
```

commit

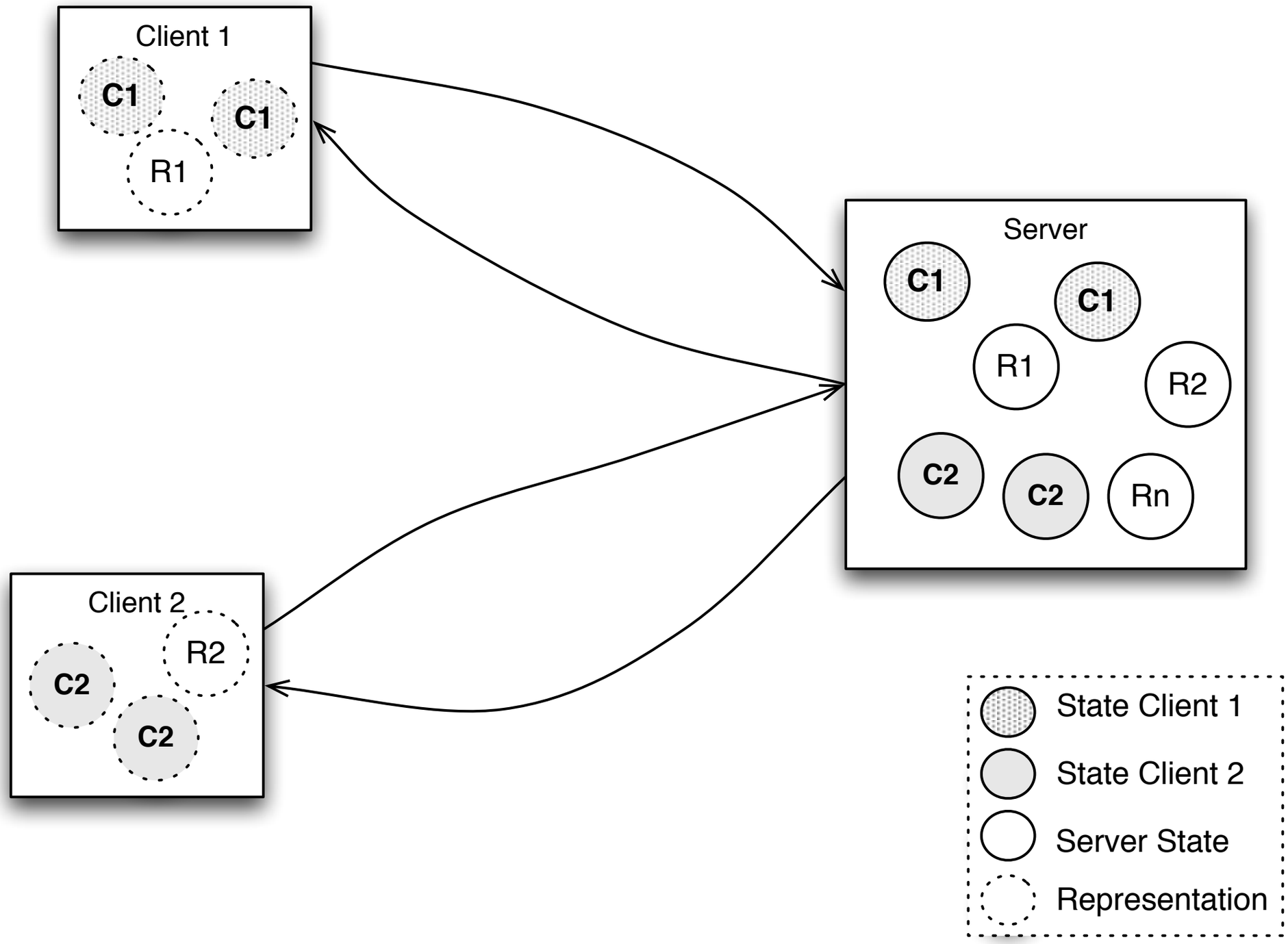
```
→PUT /registration-4711/status↵
   <status>finalized</status>
```

Stateful Communication

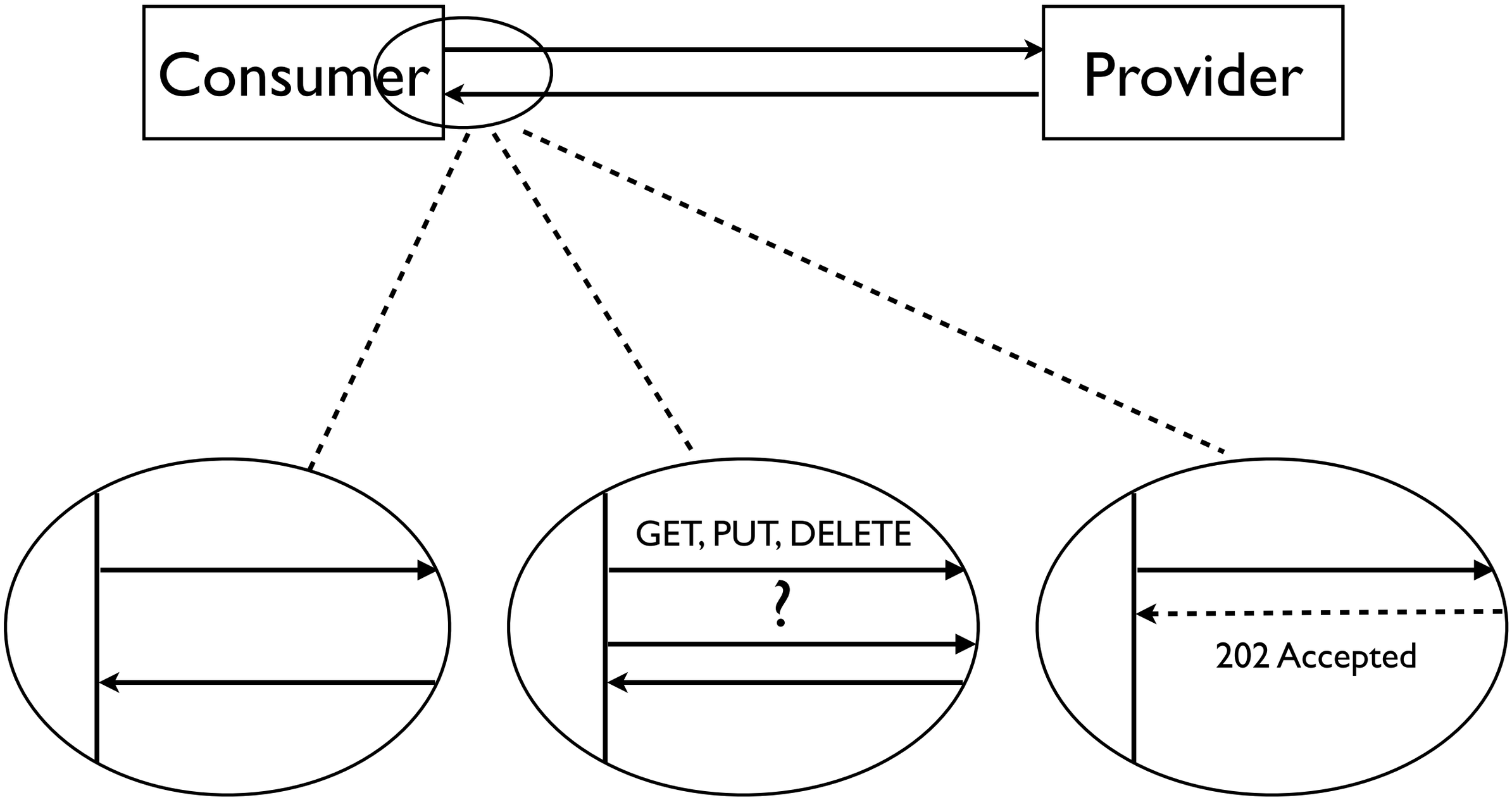
Turn session state ...



… into client or resource state



Reliable Messaging

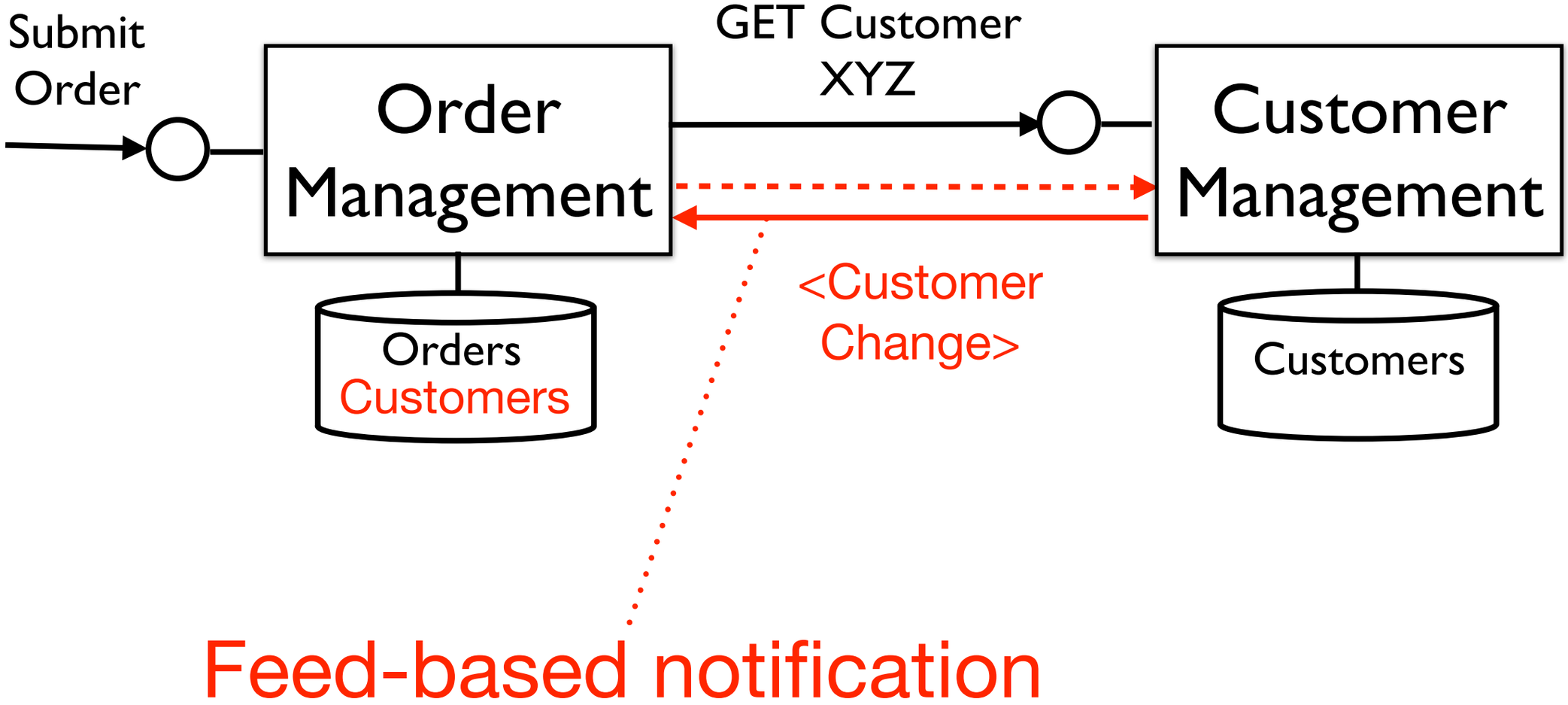


Success/
Error

Idempotent
Retry

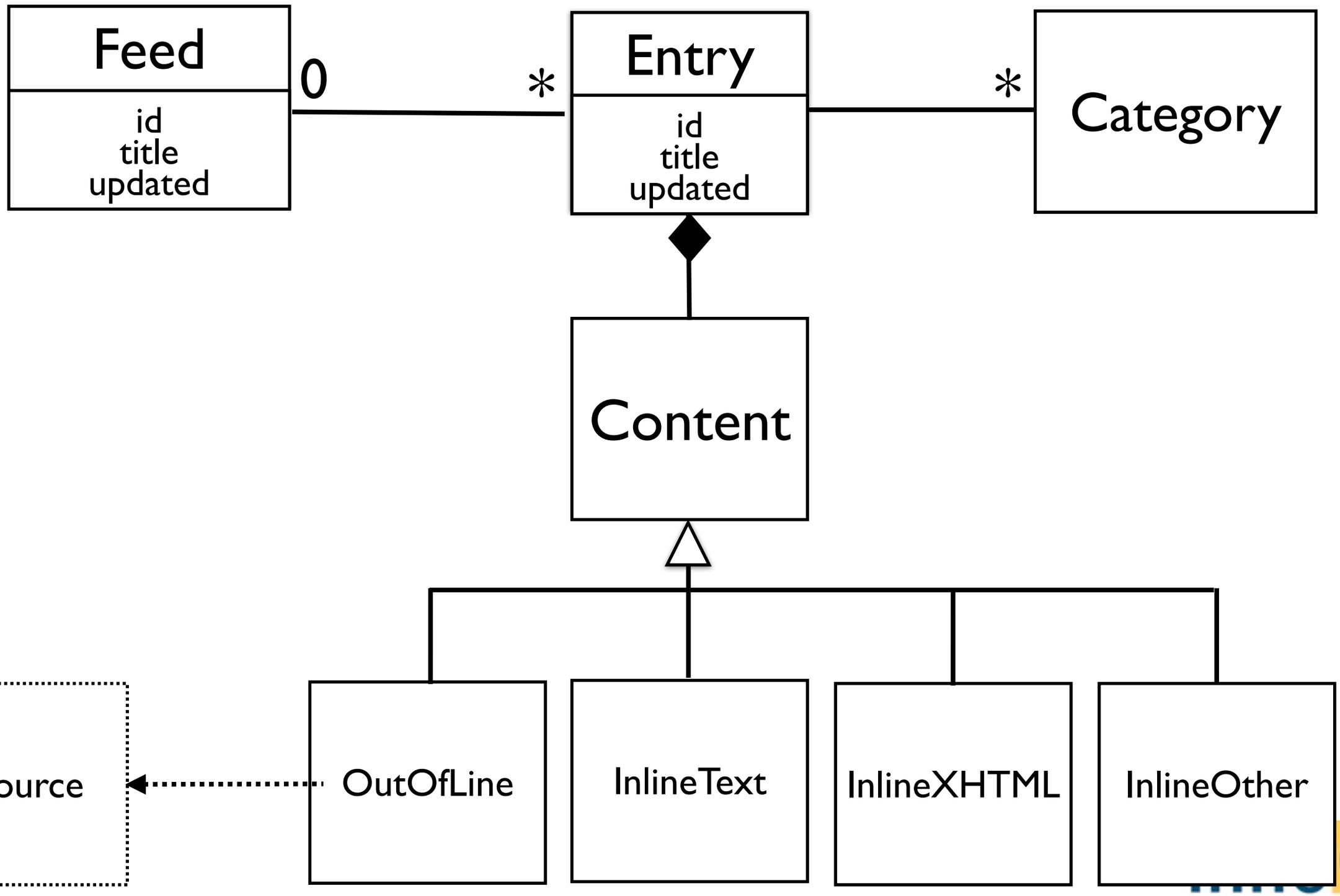
Async
Accepted

Notifications



Feed-based notification

Atom Model



REST is for services only

REST as the Web's Architectural Style

1991	HTTP 0.9
1996	HTTP 1.0
1997	HTTP 1.1 (RFC 2068)
1999	HTTP 1.1 (RFC 2616)
2000	REST
2000	SOAP/1.1

Browsers

Command line clients

Crawlers

Proxies

Servers

ROCA

http://roca-style.org/

REST is about nice URLs

What makes a URI “RESTful”?

http://example.com/customers/getDetails?id=13

http://example.com/customers/delete?id=13

http://example.com/customers/13

http://example.com/customers/13/orders

http://example.com/orders/4711/customer

There is no such thing as a “RESTful URI”

`http`	`://`	`example.com`	`/customers/delete`	`?`	`id=13`	
Scheme		**Host**	**Path**		**Param**	

Opaque ID

Why you shouldn’t care about URIs

```
<customer>
  <...>
  <orders href='http://example/customer/3'>
</customer>
```

Hypermedia context

REST requires more client-side logic

REST = CRUD

Minor differences

Create **POST**

Read **GET**

Update **PUT**

Delete **DELETE**

- When used for creation, server decides about URI
- Can also invoke arbitrary processing

- Can also be used for creation with known URI
- Not to be used for partial updates, idempotent

Major difference

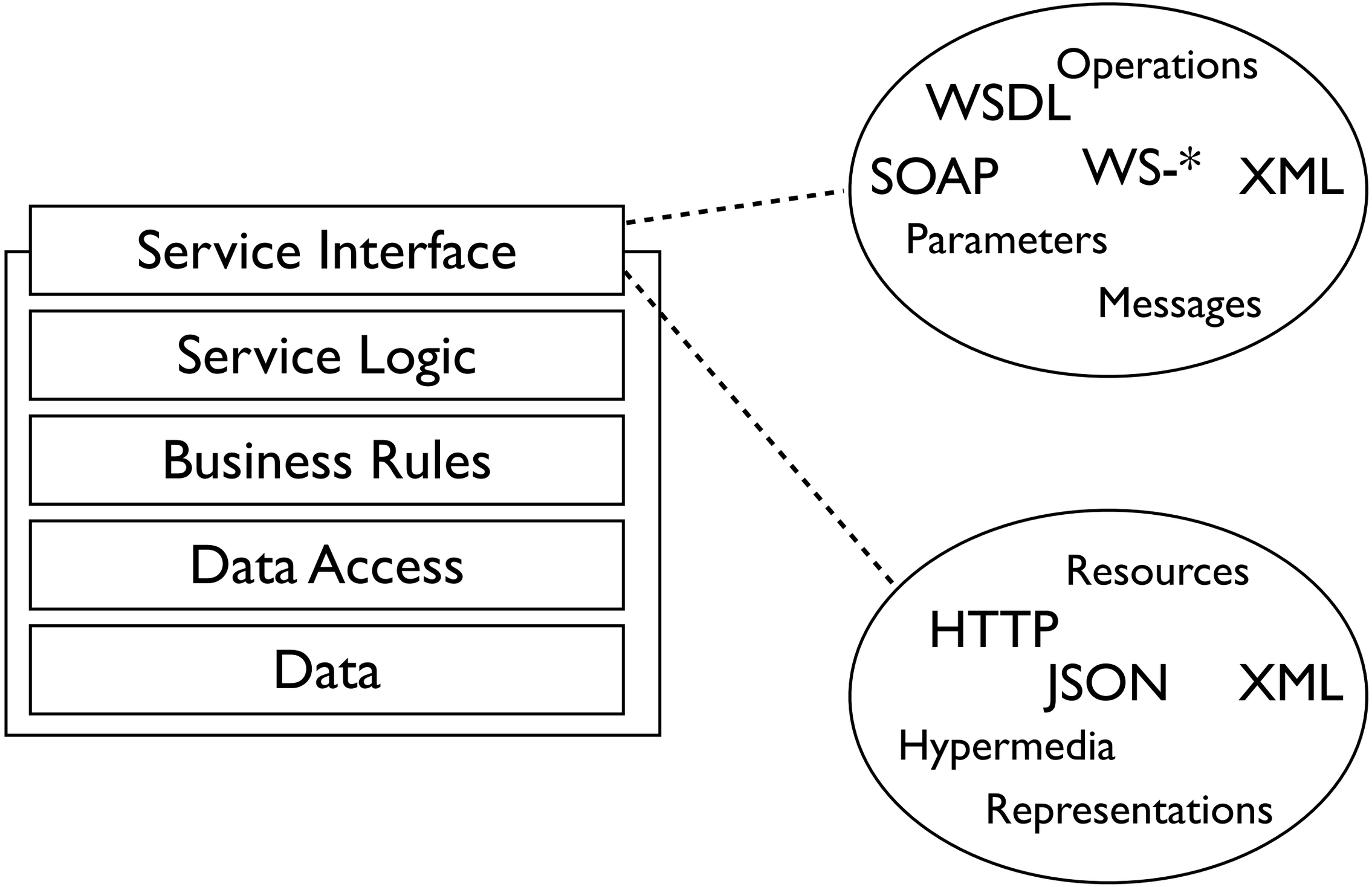
Create, Read, Update, Delete

- Operations on data
- Pure storage; business logic in caller

GET, PUT, POST, DELETE
(+ Representations + URIs + Hypermedia)

- Different interface style
- No change in logic responsibilities

innoQ



REST = URI patterns + GET, PUT, POST, DELETE

CLOSE, BUT NO

URI	Method	Meaning
http://ex.org/v1/customers	POST	create new customer
http://ex.org/v1/customers/{id}	GET	get customer details
http://ex.org/v1/customers{id}/orders	GET	get list of customer's details
...	...	...

Documented URIs become APIs

Versions in IDs cause change without reason

Inflexible assumptions about server details

Step 1: Service Documents

Document with links to “entry point” resources

Can be consumer-specific

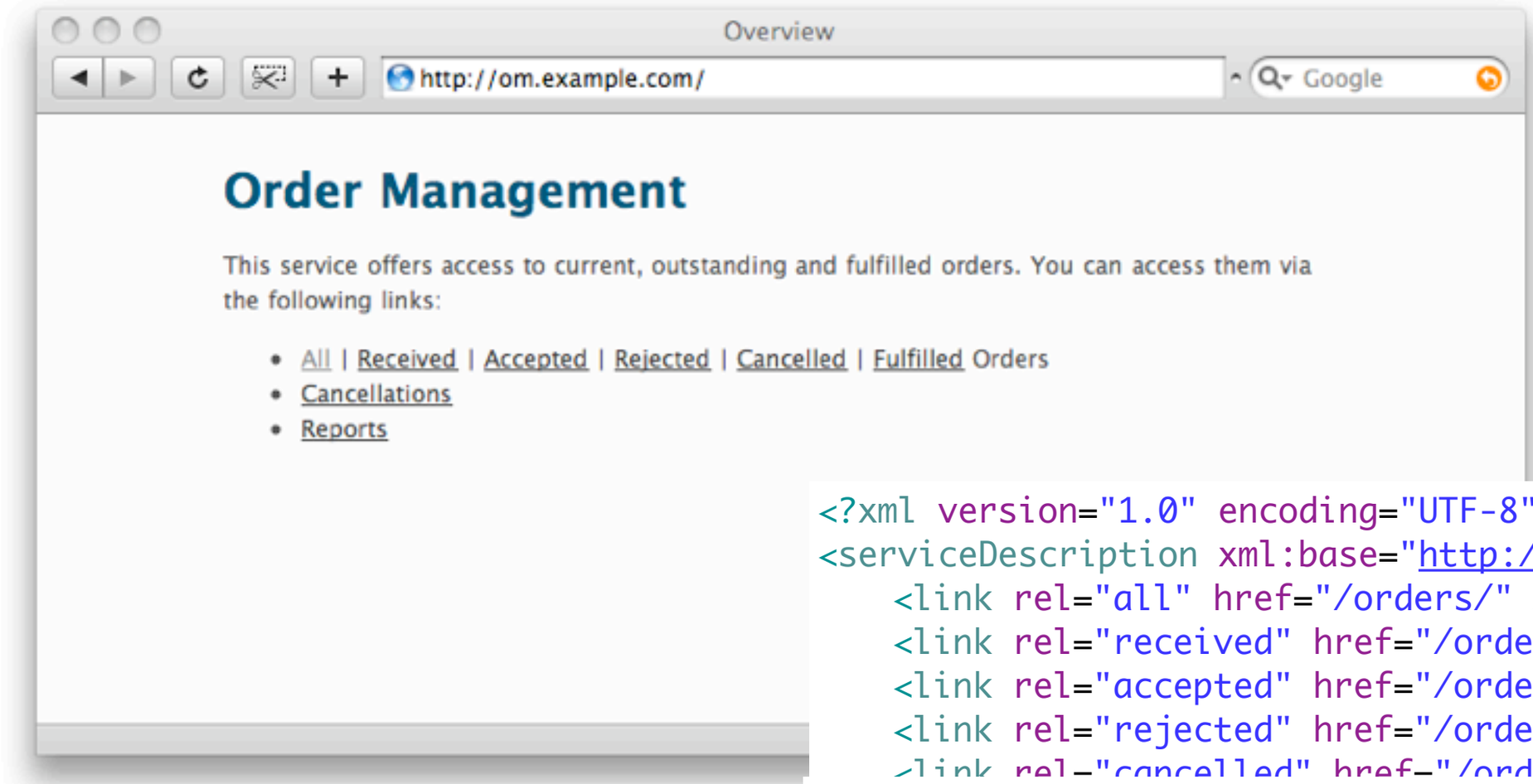
Additional “cheap” decoupling

Federated if necessary

Service documents

```
<services>
 <service id="lookup" href="http://..."/>
 <service id="crm" href="http://..."/>
 <service id="accounting" href="http://..."/>
</services>
```

Example




```
<?xml version="1.0" encoding="UTF-8"?>
<serviceDescription xml:base="http://om.example.com">
    <link rel="all" href="/orders/" />
    <link rel="received" href="/orders/received/" />
    <link rel="accepted" href="/orders/accepted/" />
    <link rel="rejected" href="/orders/rejected/" />
    <link rel="cancelled" href="/orders/cancelled/" />
    <link rel="fulfilled" href="http://om.archive.com/orders/"
    <link rel="cancellations" href="/cancellations/" />
    <link rel="reports" href="/reports/" />
</serviceDescription>
```

Link styles

```
<service id="lookup" href="http://..."/>

<link rel="service"
      name="lookup" href="http://..."/>

<link rel="lookup-service" href="http://..."/>
```

Step 2: Resource Links

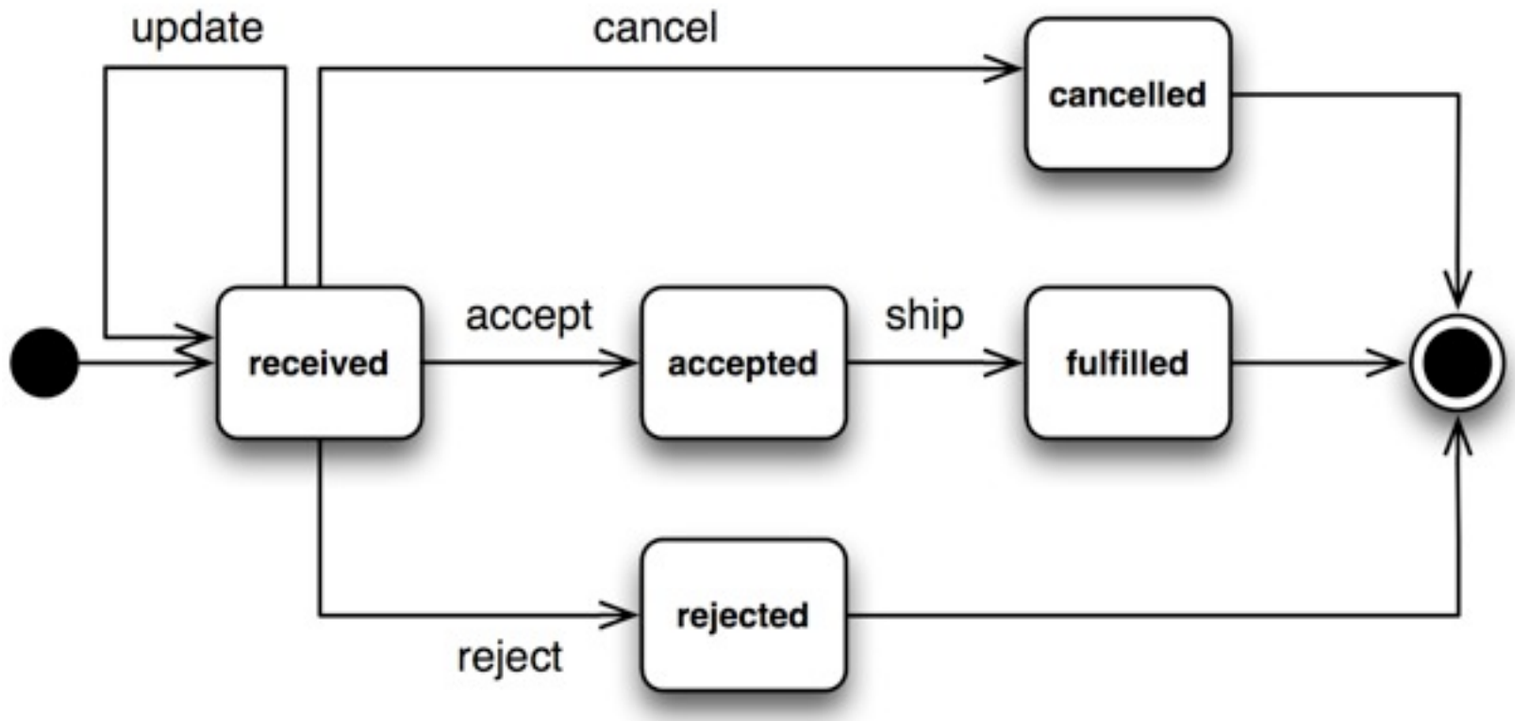
Inherited from your domain model

Links between collection- and primary resources

Links for self-references

Make even implicit relationships explicit to prevent client-side assumptions

Step 3: State Transition Links



Determine the possible client actions

Distinction from resource links is leaky since every link acts as state transition

“Named” Links

Resource relationships in representations

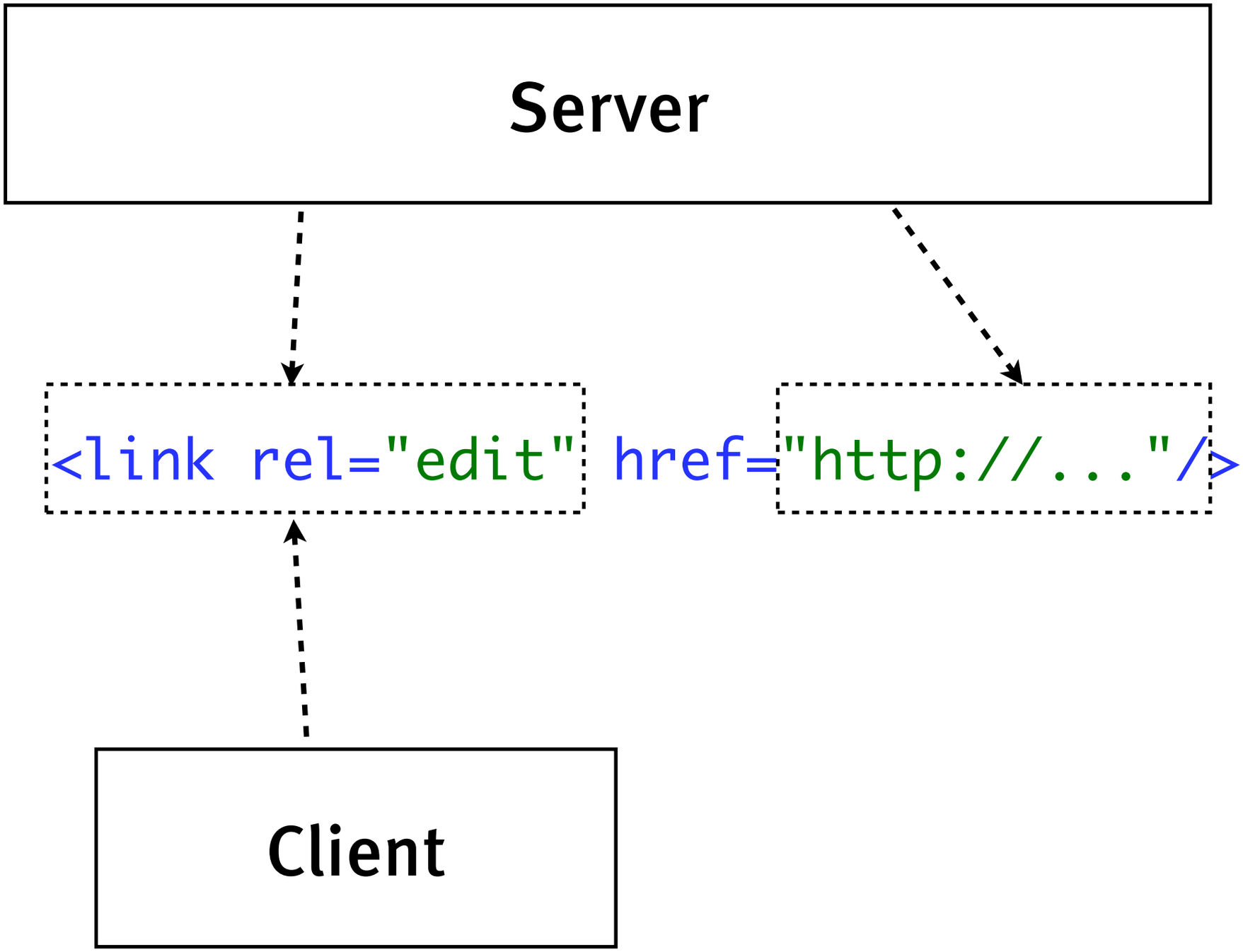
Independence from URI design

Possible “De-Co-Location”

Link Relations

```
<?xml version="1.0" encoding="UTF-8"?>
<order xml:base="http://om.example.com">
   <link rel="self" href="/orders/123"
         type="application/vnd.example.com-ordermgr+xml" /
      >
   <state>received</state>
   <link rel="payment" href="https://paypal/" />
   <link rel="cancel" href="/cancellations/" />
   <!-- ... -->
</order>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<orders xmlns="http://example.com/schemas/ordermanagement"
   xml:base="http://om.example.com">
   <link rel="self" href="/orders/?page=3" />
   <link rel="prev" href="/orders/?page=2" />
   <link rel="next" href="/orders/?page=4" />
   <!-- ... -->
</orders>
```



Summary

REST is quickly becoming mainstream

Many things that claim to be REST are not

Most common pattern today: REST without Hypermedia, a.k.a. “Web without links”

Thank you!

Q&A

Stefan Tilkov
stefan.tilkov@innoq.com
http://www.innoq.com/blog/st/
@stilkov
Phone: +49 170 471 2625

innoQ Deutschland GmbH
Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0
http://www.innoq.com

innoQ Schweiz GmbH
Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116
info@innoq.com