REST: Slightly more than an Introduction

Stefan Tilkov, @stilkov
JUG Ostfalen, AutoUni 2013-03-04
REpresentational State Transfer
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

<table>
<thead>
<tr>
<th>Identification of resources</th>
<th>Resource manipulation through representations</th>
<th>Hypermedia as the engine of application state</th>
<th>Self-descriptive messages</th>
</tr>
</thead>
</table>

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

```xml
<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

Example HTTP request:
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 19:36:25 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'?>
getOrderDetails()
submitApplicationData()
updateQuote()
findMatchingBid()
initiateProcess()
cancelSubscription()
listAuctions()
getUsers()
interface Resource {
    Resource(URI u)
    Response get()
    Response post(Request r)
    Response put(Request r)
    Response delete()
}

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

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

CustomerManagementService

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

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
many

Data types

many

Operations

very few
(one per service)

Instances

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

CustomerManagementService
+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()
<table>
<thead>
<tr>
<th>Function</th>
<th>API Request</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getFreeTimeSlots(Person)</code></td>
<td><code>GET /people/st/timeslots?state=free</code></td>
<td></td>
</tr>
<tr>
<td><code>rejectApplication(Application)</code></td>
<td><code>POST /rejections</code></td>
<td><code>&lt;application&gt;http://...&lt;/application&gt;</code>&lt;reason&gt;Unsuitable for us!&lt;/reason&gt;</td>
</tr>
<tr>
<td><code>performTariffCalculation(Data)</code></td>
<td><code>POST /contracts</code></td>
<td><code>Location: http://.../contracts/4711</code>&lt;tariff ref='./contracts/4711/tariff' /&gt;<code>GET /contracts/4711/tariff</code>&lt;Result&gt;</td>
</tr>
<tr>
<td><code>shipOrder(ID)</code></td>
<td><code>PUT /orders/0815/status</code></td>
<td><code>&lt;status&gt;shipped&lt;/status&gt;</code></td>
</tr>
<tr>
<td><code>shipOrder(ID) [variation]</code></td>
<td><code>POST /shipments</code></td>
<td><code>Location: http://.../shipments/4711</code></td>
</tr>
</tbody>
</table>
REST offers less standardization than SOAP/WSDL/WS-* Web services.

WRONG
(“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

UNTRUE
easy  hard

simple  complex
Transactions
Tx across services
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 ...

Client 1
- R1

Client 2
- R2

Server
- R1
- R2
- Rn

State Client 1
State Client 2
Server State
Representation
... into client or resource state
Reliable Messaging
Client

Consumer

Success/
Error

Idempotent
Retry

Provider

GET, PUT, DELETE

202 Accepted

Async
Accepted
Notifications
Feed-based notification
REST is for services only
REST as the Web’s Architectural Style

<table>
<thead>
<tr>
<th>Year</th>
<th>Version</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1991</td>
<td>HTTP 0.9</td>
<td></td>
</tr>
<tr>
<td>1996</td>
<td>HTTP 1.0</td>
<td>Browsers</td>
</tr>
<tr>
<td>1997</td>
<td>HTTP 1.1 (RFC 2068)</td>
<td>Command line clients</td>
</tr>
<tr>
<td>1999</td>
<td>HTTP 1.1 (RFC 2616)</td>
<td>Crawlers</td>
</tr>
<tr>
<td>2000</td>
<td>REST</td>
<td>Proxies</td>
</tr>
<tr>
<td>2000</td>
<td>SOAP/1.1</td>
<td>Servers</td>
</tr>
</tbody>
</table>
ROCA

http://roca-style.org/
REST is about nice URLs

IT'S NOT
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”

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Host</th>
<th>Path</th>
<th>Opaque ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>http://</td>
<td>example.com</td>
<td>/customers/delete?id=13</td>
<td></td>
</tr>
</tbody>
</table>

© 2013 innoQ Deutschland GmbH
Why you shouldn’t care about URIs

```xml
<customer>
  ...
  <orders href='http://example.com/customers/13/orders'>
    ...
  </orders>
</customer>
```

Hypermedia context
REST requires more client-side logic

NOT TRUE
REST = CRUD

NOT AT ALL
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
REST = URI patterns +
GET, PUT,
POST, DELETE

CLOSE, BUT NO
<table>
<thead>
<tr>
<th>URI</th>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://ex.org/v1/customers">http://ex.org/v1/customers</a></td>
<td>POST</td>
<td>create new customer</td>
</tr>
<tr>
<td><a href="http://ex.org/v1/customers/%7Bid%7D">http://ex.org/v1/customers/{id}</a></td>
<td>GET</td>
<td>get customer details</td>
</tr>
<tr>
<td><a href="http://ex.org/v1/customers%7Bid%7D/orders">http://ex.org/v1/customers{id}/orders</a></td>
<td>GET</td>
<td>get list of customer’s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>details</td>
</tr>
</tbody>
</table>

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

Order Management

This service offers access to current, outstanding and fulfilled orders. You can access them via the following links:

- All
- Received
- Accepted
- Rejected
- Cancelled
- Fulfilled
- Orders
- Cancellations
- Reports

```xml
<?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>
<link rel="edit" href="http://..."/>

Client

Server
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