REST web services

Paulo Gandra de Sousa

@pagsousa

Disclaimer

Material in this presentation is based on:

- "A Brief Introduction to REST", Stefan Tilkov
- "REST Anti-Patterns", Stefan Tilkov
- "Addressing Doubts about REST", Stefan Tilkov
- "A Guide to Designing and Building RESTful Web Services with WCF 3.5", Aaron Skonnard

INTRODUCTION

REST

- **RE**presentational **S**tate **T**ransfer
  - Roy Fielding's PhD thesis (2000)

- Two main ideas:
  1. Everything is a resource
  2. Each resource has a Uniform interface

REST is an "architectural style"

REST/HTTP is
the most common instantiation

Key principles

1. Give every "thing" an ID
2. Use standard methods
3. Provide multiple representations
4. Communicate statelessly
5. Link things together

1) Give every "thing" an ID

- Even non-usual things
  - e.g., process, process step
- Individual itens …
  - http://example.com/customers/1234
  - http://example.com/orders/2007/10/776654
  - http://example.com/products/4554
  - http://example.com/processes/salary-increase-234
- … and collections
  - http://example.com/orders/2007/11
  - http://example.com/products?color=green

URIs are **opaque**!

2) Use standard methods

- Uniform interface
  - Common set of methods
- Well defined semantics
  - E.g., Standard HTTP verbs and response codes

Uniform interface

Method	Description	Safe	Idempotent
GET	Requests a specific representation of a resource	Yes	Yes
PUT	Create or update a resource with the supplied representation	No	Yes
DELETE	Deletes the specified resource	No	Yes
POST	Submits data to be processed by the identified resource	No	No
HEAD	Similar to GET but only retrieves headers and not the body	Yes	Yes
OPTIONS	Returns the methods supported by the identified resource	Yes	Yes

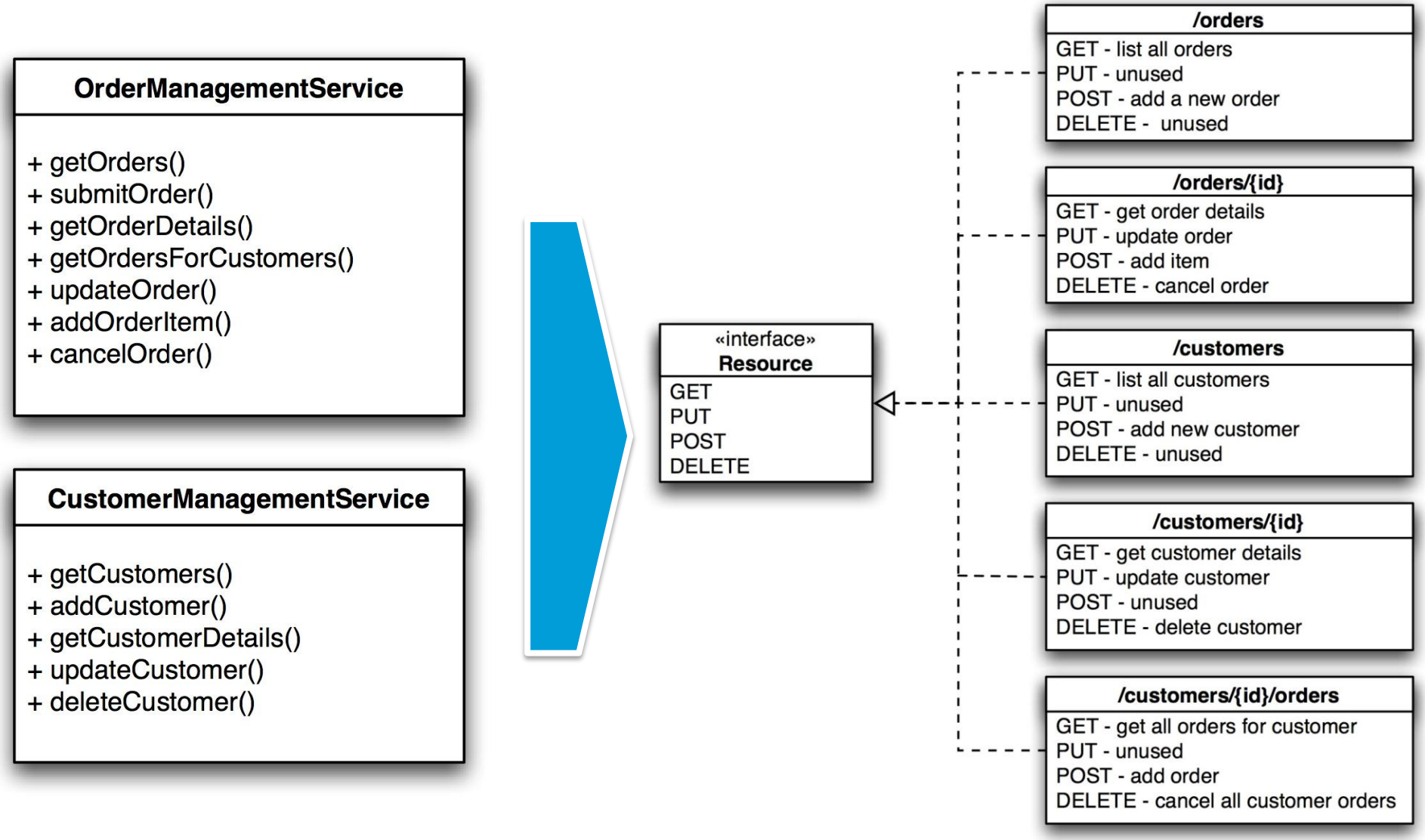
http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9

Uniform interface

Method	**Collection URI,** such as http://expl.com/customers/	**Element URI,** such as http://expl.com/customers/142
GET	List the URIs and perhaps other details of the collection's members.	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.
PUT	Replace the entire collection with another collection.	Update the addressed member of the collection, or if it doesn't exist, create it.
POST	Create a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.	Treat the addressed member as a collection in its own right and create a new entry in it.
DELETE	Delete the entire collection.	Delete the addressed member of the collection.

Source: Richardson, Leonard; Ruby, Sam (2007), *RESTful Web Services*, O'Reilly

RPC → REST


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

Response codes

Status Range	Description	Examples
100	Informational	100 Continue
200	Successful	200 OK 201 Created 202 Accepted
300	Redirection	301 Moved Permanently 304 Not Modified
400	Client error	400 Bad Request 401 Unauthorized 402 Payment Required 404 Not Found 405 Method Not Allowed 409 Conflict
500	Server error	500 Internal Server Error 501 Not Implemented

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

3) Multiple representations

XML	JSON	YAML
XHTML	Atom	…

3) Multiple representations

- Use HTTP content negotiation
  - `Accept:` header
  - Use standard MIME formats as much as possible

- Example #1:

```
GET /customers/1234 HTTP/1.1
Host: example.com
Accept: application/vnd.mycompany.customer+xml
```

- Example #2:

```
GET /customers/1234 HTTP/1.1
Host: example.com
Accept: text/x-vcard
```

4) Communicate statelessly

Resource state

or

Client state

No server session

4) Communicate statelessly

```
$> [Client:] Show me your products
[Server:] Here's a list of all the products
$> I'd like to buy 1 of http://ex.org/product/X, I am
  "John"/"Password"
I've added 1 of http://ex.org/product/X to
  http://ex.org/users/john/basket
$> I'd like to buy 1 of http://ex.org/product/Y, I am
  "John"/"Password"
I've added 1 of http://ex.org/product/Y to
  http://ex.org/users/john/basket
$> I don't want http://ex.org/product/X, remove it, I am
  "John"/"Password"
I've removed http://ex.org/product/X from
  http://ex.org/users/john/basket
$> Okay I'm done, username/password is "John"/"Password"
Here is the total cost of the items in
  http://ex.org/users/john/basket
```

http://www.peej.co.uk/articles/no-sessions.html

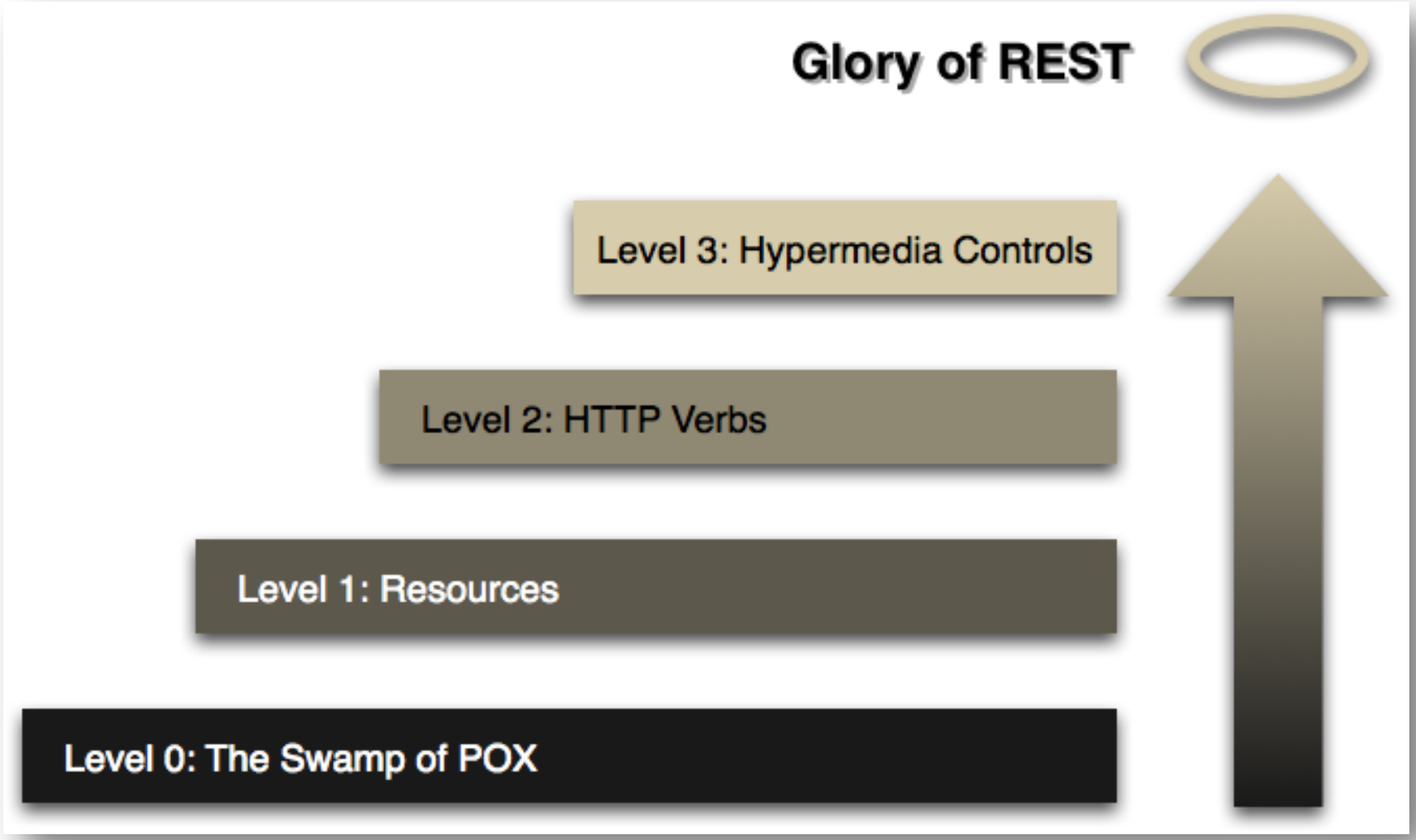
5) Link things together

Self URI

```
<order rel='self'
       href='http://example.com/orders/9876'>
   <amount>23</amount>
   <product
      href='http://example.com/products/4554' />
   <customer
      href='http://example.com/customers/1234'/>
   <dispatcher
      href='http://dispatchers.org/members/2258'/>
</order>
```

External URI

Richardson Maturity Model



http://martinfowler.com/articles/richardsonMaturityModel.html

Hipermedia as the engine of application state *

* HATEOAS

Hypermedia controls

- The resource provides info about possible "actions" to follow
- Well know semantics
- Atom:link rel
  - See IANA registry http://www.iana.org/assignments/link-relations/link-relations.xml

HATEOAS example

- Place an order

```
PUT /order HTTP/1.1
Host: http://central-cafe.com
Content: application/xml
<order> <drink>latte</drink> </order>
```

- Response

```
201 Created
Location: http://central-cafe.com/orders/1234
Content: application/xml
<order> <drink>latte</drink> <cost>3.0</cost>
<link rel="http://cafe.org/cancel"
      uri="http://central-cafe.com/order/1234"/>
<link rel="http://cafe.org/payment"
      uri="http://central-cafe.com/payment/order/1234"/>
</order>
```

Allowed actions

LEVERAGE THE WEB/HTTP

Look before you leap

- `OPTIONS` will return currently acceptable actions on a resource

- Request

```
OPTIONS /order/1234 HTTP 1.1
Host: cafe.org
```

- Response

```
200 OK
Allow: GET, PUT
```

PUT is allowed; e.g., order is not yet processed

Authentication

- Leverage HTTP authentication
- For actions that need authentication
  - The Server returns
    - `401 Unauthorized`
    - `WWW-Authenticate` header
  - The Client uses `Authorization` header

Cache

- HTTP cache is powerfull and complex
  - … and time-proven
- Performance
- Scalability

Content

- MIME-types
- Content negotiation

Concurrency

- ETags

Bookmark Management

EXAMPLE

Bookmark management

- Obtain list of public bookmarks (filtered by user and/or subject)
- Obtain list of a user's private bookmarks (filtered by subject)
- Create a user account
- Add a public/private bookmark

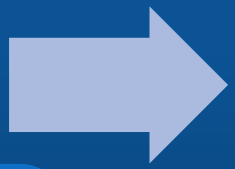
Traditional RPC-style API

Operation	Description
createUserAccount	Creates a new user account
getUserAccount	Retrieves user account details for the authenticated user
updateUserAccount	Updates user account details for the authenticated user
deleteUserAccount	Deletes the authenticated user's account
getUserProfile	Retrieves a specific user's public profile information
createBookmark	Creates a new bookmark for the authenticated user
updateBookmark	Updates an existing bookmark for the authenticated user
deleteBookmark	Deletes one of the authenticated user's bookmarks
getBookmark	Retrieves a specific bookmark (anyone can retrieve a public bookmark; only authenticated users can retrieve a private bookmark)
getUserBookmarks	Retrieves the user's private bookmarks, allows filtering by tags
getUserPublicBookmarks	Retrieves the user's public bookmarks, allows filtering by tags
getPublicBookmarks	Retrieves all public bookmarks, allows filtering by tags

From verbs to nouns

RPC API

- Verb/Action
- e.g., getBookmark
- e.g., createUser



REST API

- Resource/Noum
- e.g., Bookmark
- e.g., User

Resources in the sample service

- An individual user account
- A specific user's public profile
- An individual bookmark
- A user's collection of private bookmarks
- A user's collection of public bookmarks
- The collection of all public bookmarks

Designing the URI

- Create URI templates for accessing the resources
  - Start with service's base URI
  - E.g., http://bmark.org/
- May use path components to dissambiguate
- May use query string for further semantics

Public bookmarks

- http://bmark.org/public
- Filter by subject
  - http://bmark.org/public**?tag={subject}**
- Filter by user
  - http://bmark.org/public**/{username}**
- Combining filters
  - http://bmark.org/public**/{username} ?tag={subject}**

User accounts

- Specific user account
  - http://bookmark.org**/users/{username}**
- Specific user profile
  - http://bookmark.org/users/{username}/**profile**

User bookmarks

- http://bmark.org/**users/{username}/bookmarks**
- Filter by subject
  - http://bmark.org/users/{username}/bookmarks **?tag={subject}**
- Individual bookmark
  - http://bmark.org/users/{username}/bookmarks**/{id}**

REST API

Method	URI Template	Equivalent RPC operation
PUT	users/{username}	createUserAccount
GET	users/{username}	getUserAccount
PUT	users/{username}	updateUserAccount
DELETE	users/{username}	deleteUserAccount
GET	users/{username}/profile	getUserProfile
POST	users/{username}/bookmarks	createBookmark
PUT	users/{username}/bookmarks/{id}	updateBookmark
DELETE	users/{username}/bookmarks/{id}	deleteBookmark
GET	users/{username}/bookmarks/{id}	getBookmark
GET	users/{username}/bookmarks?tag={tag}	getUserBookmarks
GET	{username}?tag={tag}	getUserPublicBookmarks
GET	?tag={tag}	getPublicBookmarks

REST API

- Resources
- URI
- Verb
- Resource representation
- Response codes

Create user account

- Request

```
PUT /users/skonard HTTP/1.1
Host: bmark.org
<User>
    <Email>aaron@pluralsight.com</Email>
    <Name>Aaron Skonnard</Name>
</User>
```

- Response

```
201 Created
Location: http://bmark.org/users/skonard
```

Get user account

- Request

```
GET /users/skonard HTTP/1.1
Host: bmark.org
Accept: application/vnd.contoso.user+xml
```

- Response

```
200 Ok
<User>
  <Email>aaron@pluralsight.com</Email>
  <Name>Aaron Skonnard</Name>
  <Bookmarks
ref='http://bmark.org/users/skonard/bookmarks'/>
  <Id>http://bmark.org/users/skonard</Id>
</User>
```

Provide links to other resources

Add a bookmark

- Request

```
POST /users/skonard/bookmarks HTTP/1.1
Host: bmark.org
<Bookmark>
   <Public>true</Public>
   <Tags>REST,WCF</Tags>
   <Title>Aaron's Blog</Title>
   <Url>http://pluralsight.com/aaron</Url>
</Bookmark>
```

- Response

```
201 Created
Location: http://bmark.org/users/skonard/bookmarks/13
```

List of bookmarks

```
<Bookmarks>
   <Bookmark>
      <Id>http://bmark.org/users/skonnard/bookmarks/13</Id>
      <LastModified>2008-03-12T00:00:00</LastModified>
      <Public>true</Public>
      <Tags>REST,WCF</Tags>
      <Title>Aaron's Blog</Title>
      <Url>http://pluralsight.com/aaron</Url>
      <User>skonnard</User>
      <UserProfile>http://bmark.org/users/skonnard/profile
      </UserProfile>
   </Bookmark>
  <Bookmark>...</Bookmark>
  <Bookmark>...</Bookmark>
</Bookmarks>
```

Resource representation

- XML
- JSON
- ATOM

- Leveraging query string
  - E.g., http://bmark.org/public?tag={subject}**&format={fmt}**

Favour HTTP content negotiation

Resource representation

- XML

```
<User>
    <Email>aaron@pluralsight.com</Email>
    <Name>Aaron Skonnard</Name>
</User>
```

- JSON

```
{User:{Email:'aaron@pluralsight.com', Name:'Aaron
  Skonnard'}}
```

ATOM feed for bookmarks

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Public Bookmarks</title>
  <updated>2008-09-13T18:30:02Z</updated>
  <id>http://bmark.org/feed</id>
  <entry>
    <author>
      <name>Aaron Skonnard</name>
    </author>
    <title>Aaron's Blog</title>
    <link href="http://pluralsight.com/aaron"/>
    <id>http://bmark.org/users/skonnard/bookmarks/13</id>
    <updated>2008-09-13T18:30:02Z</updated>
    <category term="REST,WCF"/>
  </entry>
  <entry>...</entry>
  <entry>...</entry>
</feed>
```

COMMON DOUBTS ON REST

REST = CRUD

PUT	Create
GET	Read
POST	Update
DELETE	Delete

What about "real" business logic?

REST ≠ CRUD

1) HTTP verbs do not map 1:1 to CRUD

PUT/POST	Create
GET	Read
PUT/POST	Update
DELETE	Delete

REST ≠ CRUD

2) Calculation results are resources

- One way

```
GET /sum?a=2&b=3
```

Remember GET **demands** idempotence

- A more RESTful way

```
POST /sums
<a>2</a><b>3</b>

201 Created
Location: http://expl.com/sums/1A0BD3FT98H6
<number>5</number>
```

Result's URI

No formal contract

- The Uniform Interface is the contract
- XSD, DTDs, etc. are still available
- Resources may provide a XHTML representation with documentation and data
- WADL

How to do pub/sub?

Atom

RSS

(Pull instead of push & cached results for scalability)

How to handle assync requests?

1. `POST` the request
2. Receive a `202 Accepted` or `201 Created` with result URI
3. Pull the URI to `GET` result

How to handle assync requests?

1. `POST` the request providing a callback URL
2. Receive a `202 Accepted`
3. "Wait" for server to `POST` result's URI (and result representation) to the callback URL
4. Optionaly `GET` result

How to handle transactions?

1. Treat transaction as a resource
2. Change the transaction resource itself (possible multiple `POST` steps)
3. `PUT` to transaction to commit changes

How to circunvent the web today?

- Most Firewalls only allow GET/POST
  - Use `X-HTTP-Method-Override` header

```
POST /users/skonnard/bookmarks/123 HTTP/1.1
X-HTTP-Method-Override: DELETE
```

REST anti-patterns

COMMON MISUSES

"unRESTful" GET/POST

- URIs encode operations
  - http://expl.com/svc?**method=deleteCustomer**&id=1234

- Message bodies encode operations

```
POST /svc HTTP/1.1
Host: expl.com
<operation method='deleteCustomer' />
<Customer Id="1234" />
```

Single endpoint

Exposing implementation details

Resources ≠ database entities

URIs ≠ database IDs

Ignoring caching

- Some web frameworks disable cache by default
  - `Cache-control: no-cache`

Ignoring return codes

- There is more to the web than `200 Ok`
- HTTP rich set of response codes
  - Semantically richer conversation
  - Example:

```
201 Created
Location: http://expl.com/customers/1234
```

Forgetting hypermedia

(No links in the representations of resources)

Ignoring MIME types

- Not using standard MIME types
- Not providing different representations

CONCLUSIONS

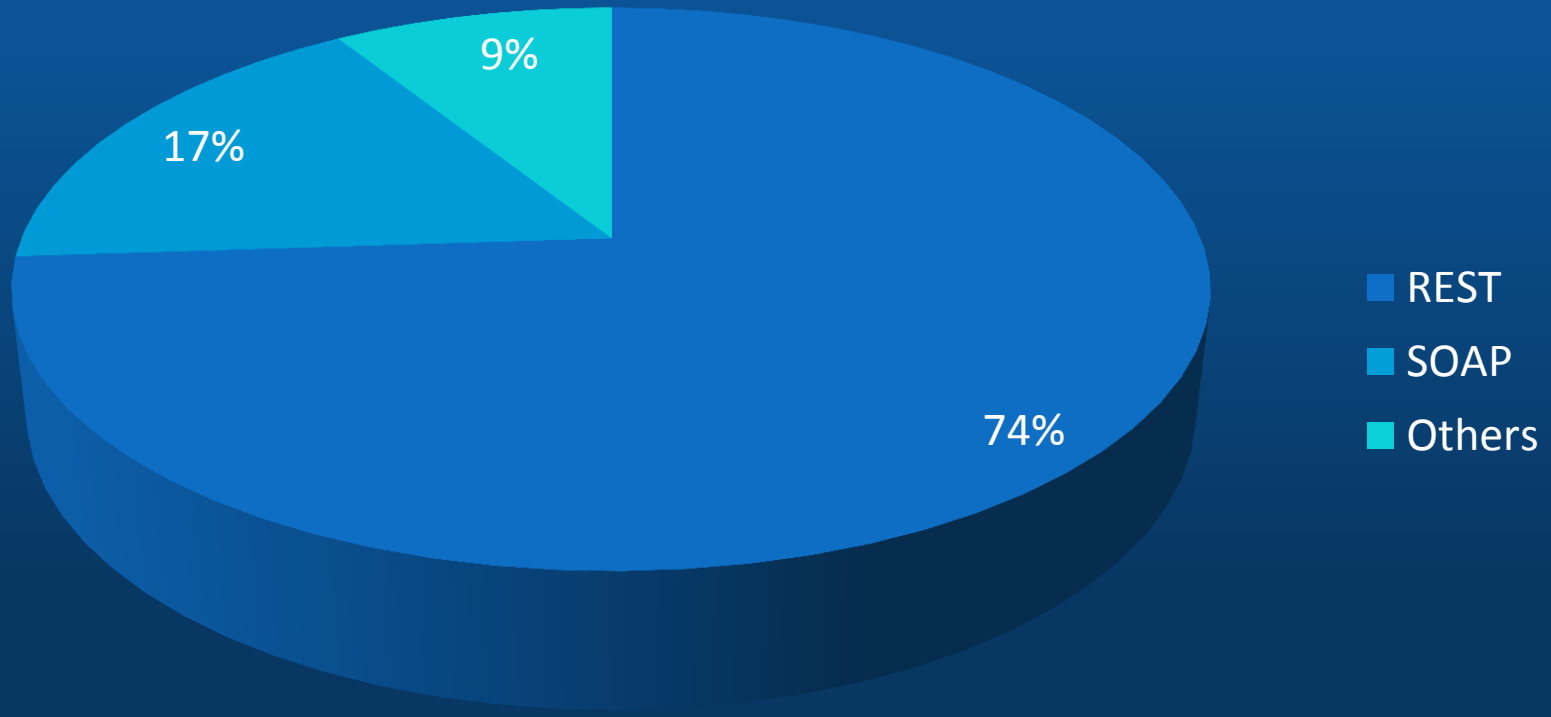
1. Follow key principles

2. Leverage web/HTTP

3. Avoid anti-paterns

Is REST really used?

(source: ProgrammableWeb @ 2010.10.29)



Added value to the web

SOAP/WS-*	REST
• 1 URL for each endpoint	• Millions of URLs – One for each resource
• 1 method: POST	• 6 methods per resource
• Closed application-specific vocabulary – E.g., createCustomer	• Standard HTTP vocabulary – Open to every HTTP-compliant client

REST
is
SOA
done correctly*

* At least to a large chorus of experts

BIBLIOGRAPHY

- “Architectural Styles and the Design of Network-based Software Architectures”, PhD Thesis (2000), Roy Thomas Fielding. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- **REST in Practice: Hypermedia and Systems Architecture (2010) Jim Webber, Savas Parastatidis, Ian Robinson. O’Reilly Media.**
- Richardson Maturity Model, Martin fowler. http://martinfowler.com/articles/richardsonMaturityModel.html
- “A Brief Introduction to REST”, Stefan Tilkov, Dec 10, 2007. http://www.infoq.com/articles/rest-introduction
- HTTP 1.1 methods definition. http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9
- HTTP 1.1 Status code definitions. http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
- “A Guide to Designing and Building RESTful Web Services with WCF 3.5”, Aaron Skonnard, Pluralsight (October 2008). http://msdn.microsoft.com/en-us/library/dd203052.aspx
- “Addressing Doubts about REST”, Stefan Tilkov, Mar 13, 2008. http://www.infoq.com/articles/tilkov-rest-doubts
- “REST Anti-Patterns”, Stefan Tilkov, Jul 02, 2008. http://www.infoq.com/articles/rest-anti-patterns
- “How to get a cup of coffe”, Jim Webber, Savas Parastatidis & Ian Robinson. Oct 02, 2008. http://www.infoq.com/articles/webber-rest-workflow