REST & JSR 311

Stefan Tilkov, innoQ Deutschland GmbH
stefan.tilkov@innoq.com

February 2008

Contents

Stefan Tilkov

http://www.innoQ.com

stefan.tilkov@innoq.com

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

174,255 Sep unique visitors

http://www.InfoQ.com

http://www.soa-expertenwissen.de

Gernot Starke · Stefan Tilkov (Hrsg.)

SOA-Expertenwissen

Methoden, Konzepte und Praxis serviceorientierter Architekturen

dpunkt.verlag

REST vs. ...?

REST vs. SOA?

REST vs. SOAP?

REST vs. WS-*?

REST is a different approach to achieve high-level SOA goals

REST Explained in 5 Easy Steps

What is REST?

REpresentational **S**tate **T**ransfer

Described by Roy Fielding in his dissertation

One of a number of “architectural styles”

Architectural principles underlying HTTP, defined *a posteriori*

See: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

0. Prerequisite: Let’s equate “REST” with “RESTful HTTP usage” ...

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”

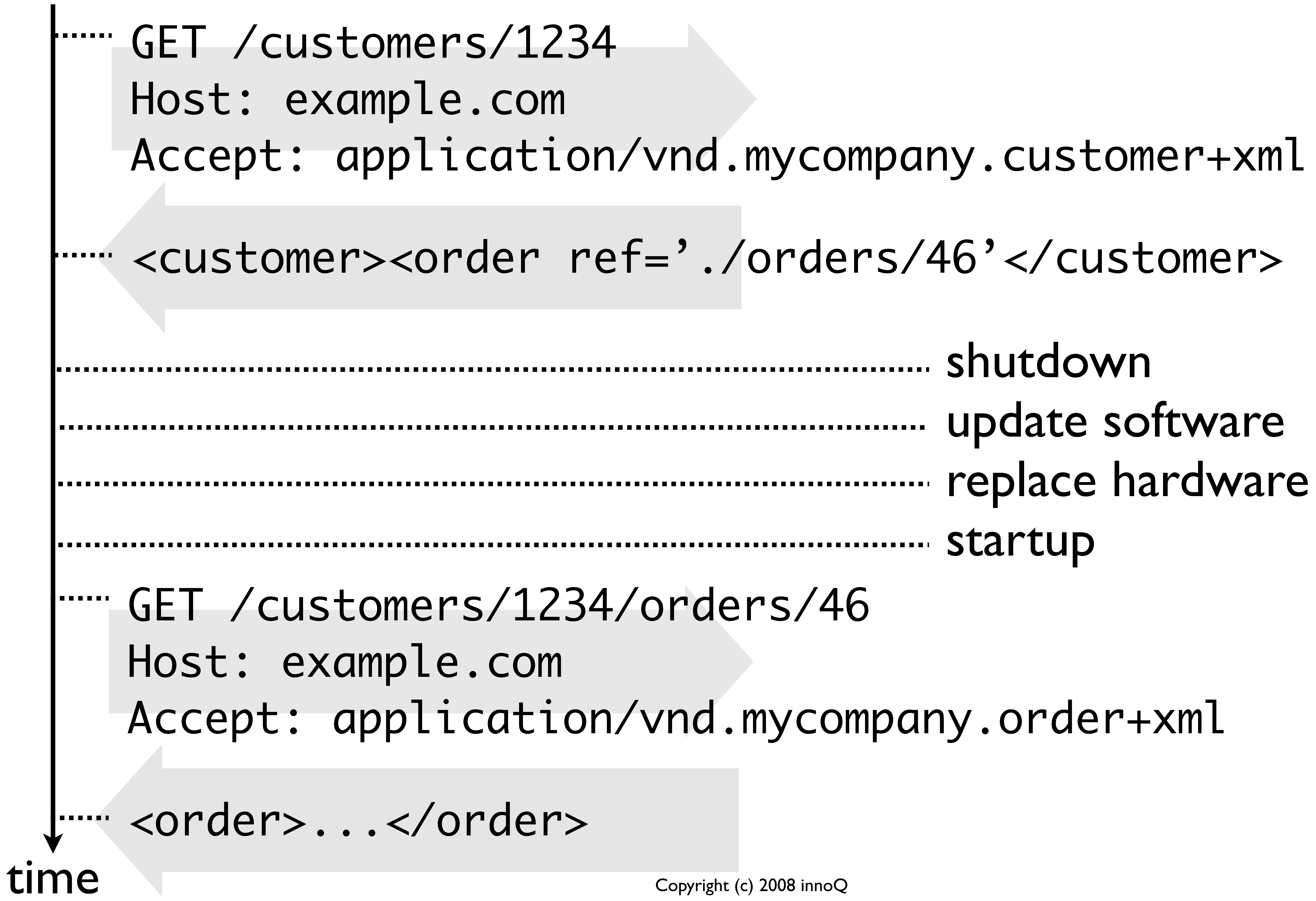
```
GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml
```

```
<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>
```

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

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

REST (Pragmatic Version)

1 Give everything an ID

2 Link things to each other

3 Use standard methods

4 Allow for multiple representations

5 Communicate Statelessly

REST (Academic Version)

1 Identifiable resources

2 Hypermedia as the engine of application state

3 Uniform interface

4 Resource representations

5 Stateless communication

Some HTTP features

Verbs (in order of popularity):

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

Standardized (& meaningful) response codes

Content negotiation

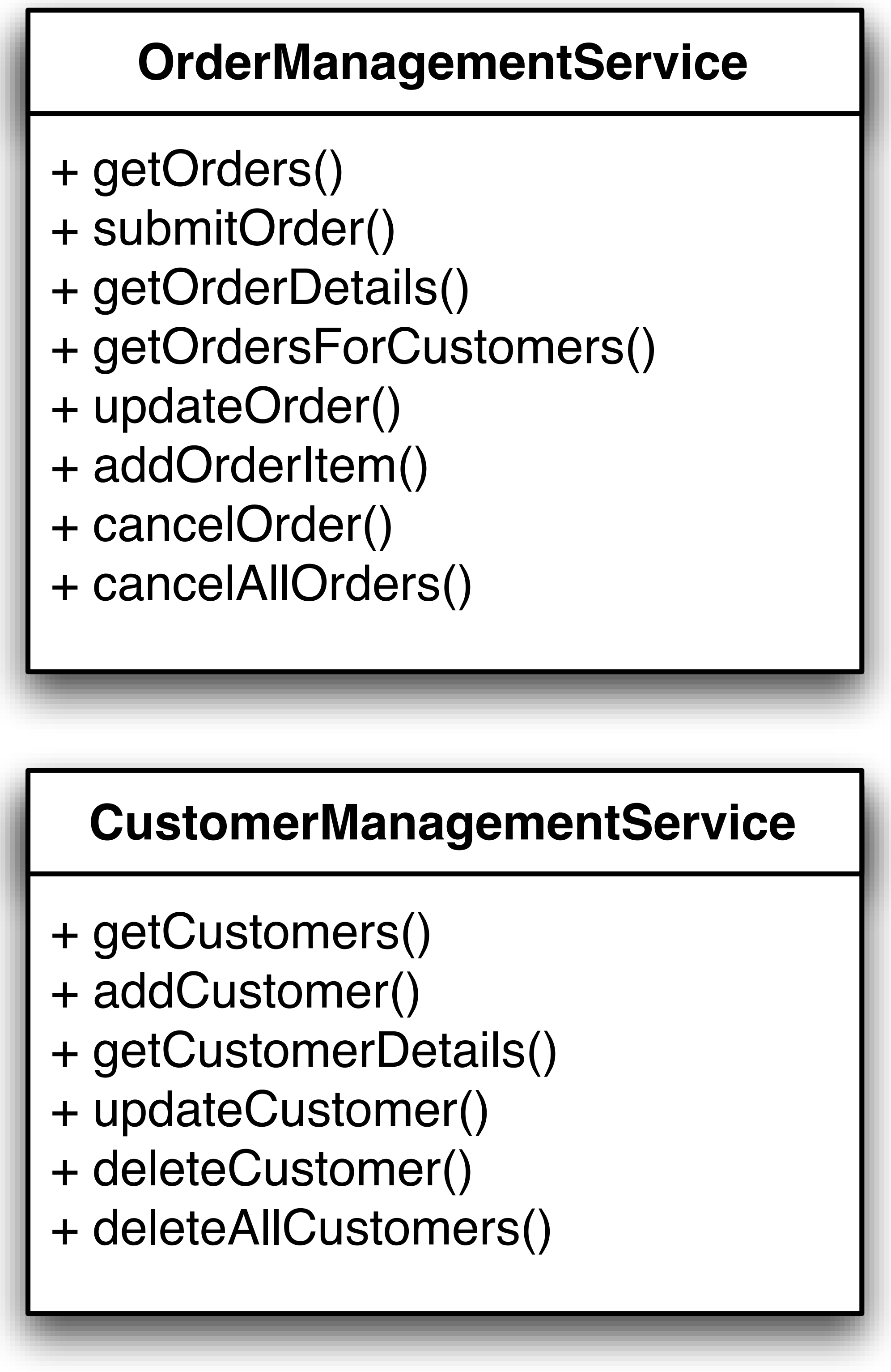
Redirection

Caching (incl. validation/expiry)

Compression

Chunking

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

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

Designing a RESTful application

Identify resources & design URIs

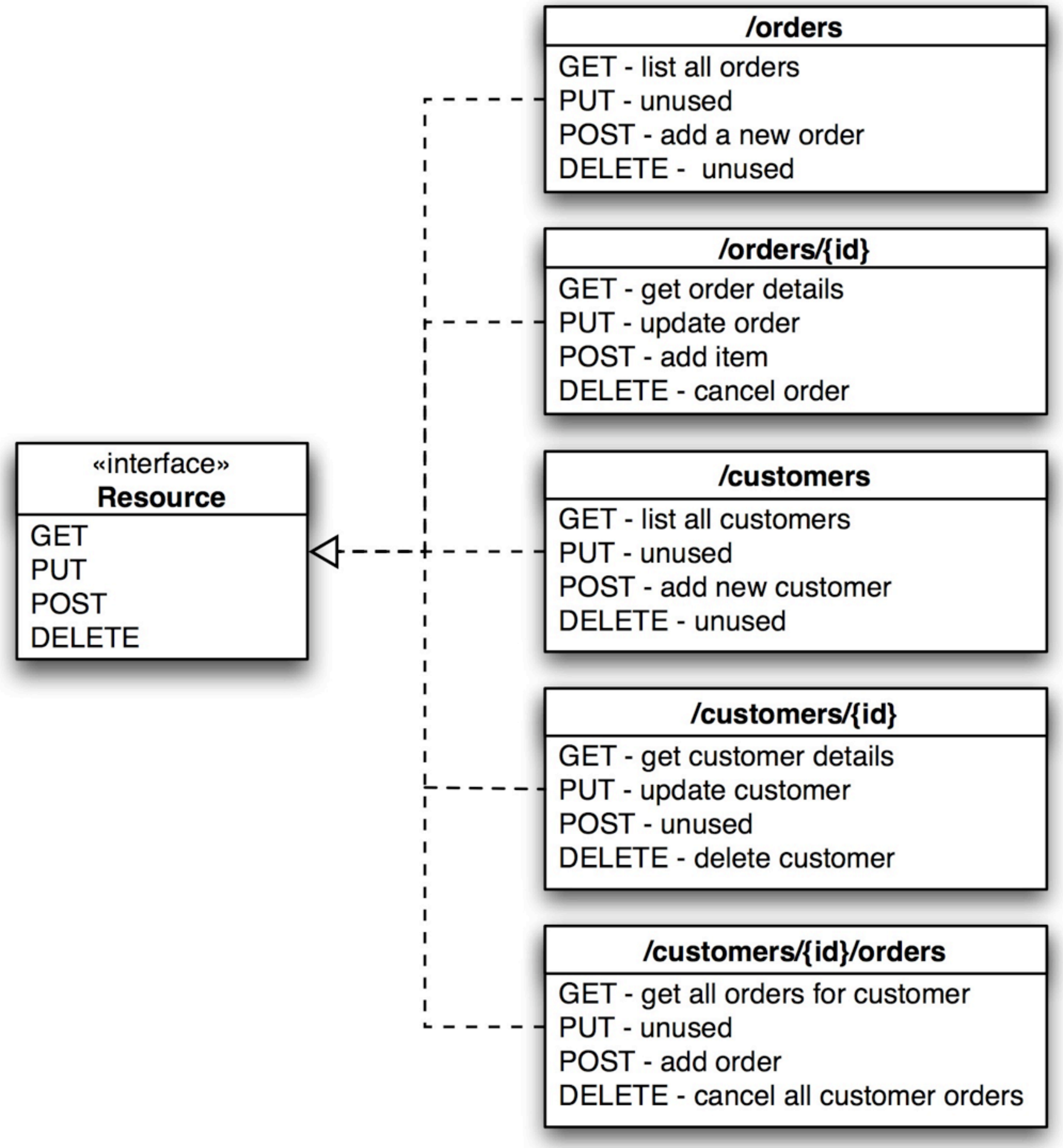
Select formats (or create new ones)

Identify method semantics

Select response codes

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

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

4-7 supported methods per resource

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

Cacheable, addressable, linkable, ...

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

Why You Should Care

WS-* Roots

The Enterprise

RPC, COM, CORBA, RMI, EJB

Transaction Systems

Controlled Environment

Top-down Approach

REST Roots

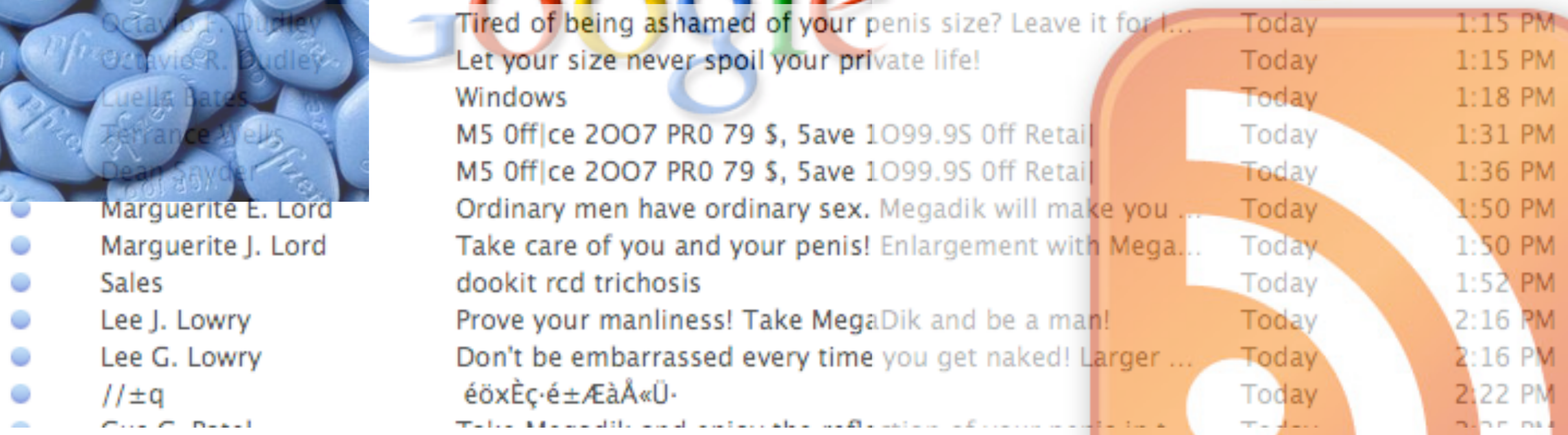
The Internet

Text formats

Wire Standards

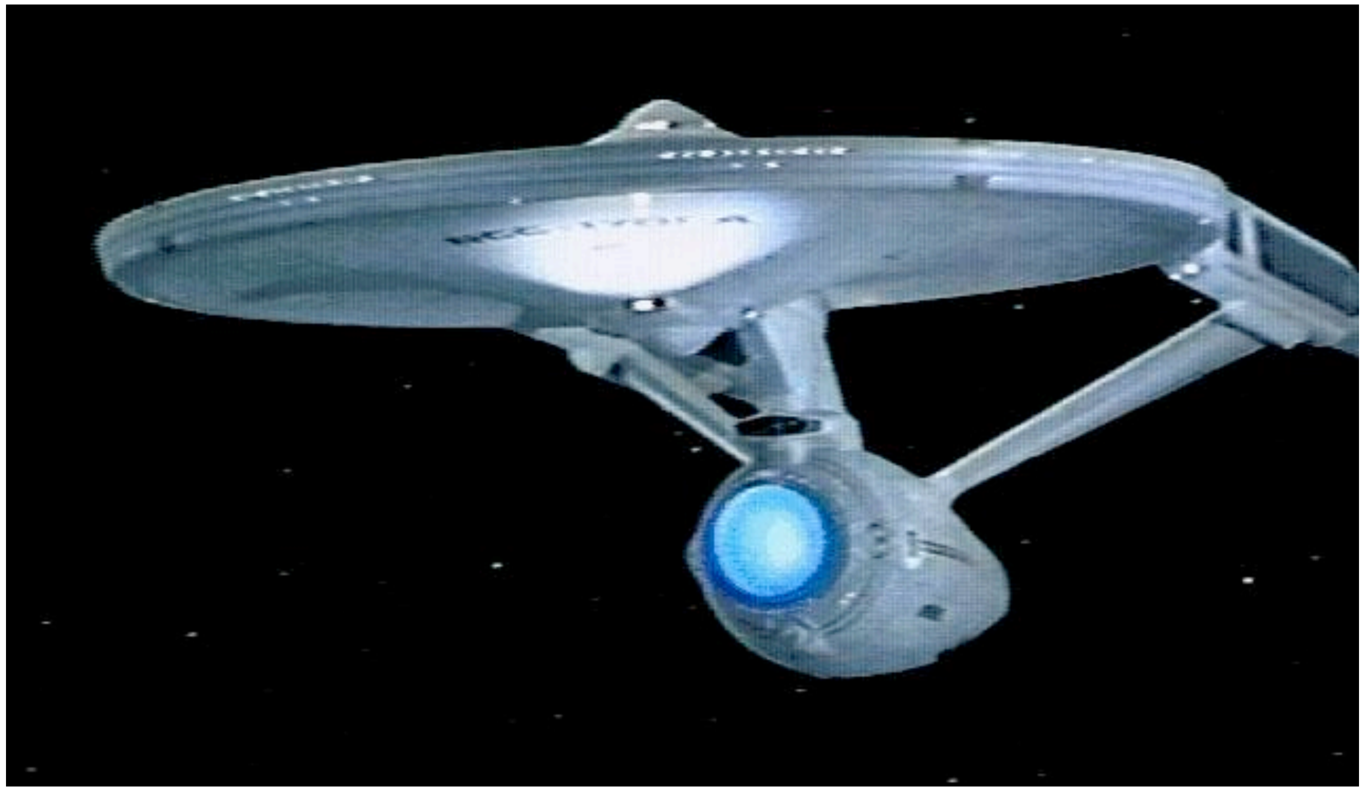
FTP, POP, SMTP

Bottom-up Approach



Octavio E. Dudley	Tired of being ashamed of your penis size? Leave it for I...	Today	1:15 PM
Octavio R. Dudley	Let your size never spoil your private life!	Today	1:15 PM
Luella Bates	Windows	Today	1:18 PM
Terrance Wells	M5 Off|ce 2OO7 PRO 79 $, 5ave 1O99.9S Off Retail	Today	1:31 PM
Dean Snyder	M5 Off|ce 2OO7 PRO 79 $, 5ave 1O99.9S Off Retail	Today	1:36 PM
Marguerite E. Lord	Ordinary men have ordinary sex. Megadik will make you ...	Today	1:50 PM
Marguerite J. Lord	Take care of you and your penis! Enlargement with Mega...	Today	1:50 PM
Sales	dookit rcd trichosis	Today	1:52 PM
Lee J. Lowry	Prove your manliness! Take MegaDik and be a man!	Today	2:16 PM
Lee G. Lowry	Don't be embarrassed every time you get naked! Larger ...	Today	2:16 PM
//±q	éöxÈç·é±ÆàÅ«Ü·	Today	2:22 PM

Internet vs. Enterprise



What’s the difference between the Internet and a typical enterprise?

Internet vs. Enterprise

One is a gigantic, uncontrollable anarchy of heterogeneous systems with varying quality that evolve independently and constantly get connected in new and unexpected ways.

The other is a worldwide, publicly accessible series of interconnected computer networks that transmit data by packet switching using the standard Internet Protocol (IP).

If web services are supposed to work on Internet scale, they should be inspired by the Web, not by Distributed Objects

JSR 311:
JAX-RS: The Java™ API for RESTful Web Services

Goals

Create a Java API for building applications that are *on* the Web *easily*

Follow REST principles and best practices

Format-independent (not only XML)

HTTP-centric (no protocol independence)

Container-independent

Status

Current status:

Feb 2007	Initiated, Expert Group formed
Oct 2007	Early Draft Review (end: Nov 23, 2007)

Original schedule:

Feb 2007	Expert group formed
Jun 2007	First expert draft
Aug 2007	Early Draft review
Nov 2007	Public Review
Jan 2008	Proposed final draft
Mar 2008	Final release.

Spec and RI

Specification available at
http://jcp.org/aboutJava/communityprocess/edr/jsr311/index.html

Jersey (reference implementation from Sun), currently at V0.6
https://jersey.dev.java.net

Approach

One class per resource “type”

Methods to handle HTTP requests

Use of Java 5 Annotations to specify

- URI Mapping
- Mapping to HTTP methods
- Mapping of URI components, HTTP headers, HTTP entities to method parameters and return types
- MIME type information

Example

/customers

GET - list all customers
PUT - unused
POST - add new customer
DELETE - delete all customers

```
import javax.ws.rs.ProduceMime;
import javax.ws.rs.UriTemplate;
import javax.ws.rs.GET;

@Path("/customers/")
public class CustomersResource {

    @GET @ProduceMime("text/plain")
    public String getAsPlainText() {
        return toString() +  "\n\n";
    }


}
```

URI Templates

URI Templates define URI strings with embedded variables

http://example.org/products/{upc}/buyers?page={page_num}

Based on Joe Gregorio's URI Templates IETF Draft (see http://bitworking.org/projects/URI-Templates/)

`@Path` annotation can be applied to classes and methods

@Path

`@Path` on a class “anchors” a class into URI space, relative to a base URI

Method-specific `@Path` is relative to the class URI

`@PathParam`, `@QueryParam`, `@MatrixParam` to access URI templates variables

@GET, @PUT, @POST, @DELETE

Specify the HTTP “verb” a method handles (GET, PUT, POST, DELETE, ...)

If not specified, verb default according to start of method name

HEAD and OPTIONS handled by implementation (unless overridden in case of HEAD)

Example

```
@Path("/helloworld/{section}")
public class HelloWorldResource {

    @GET @Path("/{id}")
    public String findBySectionAndId(
        @PathParam("section") String section,
        @PathParam("id") int id) {
        return "Hello World - section is " + section
               + ", id is " + id + "\n";
    }
}
```

http://localhost:9998/helloworld/main/23

```
Hello World - section is main, id is 23
```

Content Negotiation: @ConsumeMime, @ProduceMime

`@ConsumeMime` and `@ProduceMime` specify accepted and delivered MIME types

Can be specified on class and method level (method level overrides)

Special treatment for `MessageBodyWriter` and `MessageBodyReader` classes

Request dispatching

1. Find class and method according to
    - Actual URI and `@Path`
    - HTTP method and `@GET, @POST, @PUT, @DELETE`
    - “Content-type:” header and `@ConsumeMime`
    - “Accept:” header and `@ProduceMime`
2. Map `@UriParam`, `@QueryParam`, `@MatrixParam` parameters from URI
3. Map body (for POST and PUT) to un-annotated parameter
4. Invoke method
5. Map return value (if any)

Example

```
@Path("customers/")
public class CustomersResource {

    @GET @ProduceMime("text/plain")
    public String getAsPlainText() {
        return toString() +  "\n\n";
    }

    @GET @ProduceMime("application/vnd.innoq.customers+xml")
    public String getAsXml() {
        List<Customer> customers = Customer.findAll();
        // ...
        return elementToXmlString(root);
    }

    @POST @ConsumeMime("application/vnd.innoq.customer+xml")
    public Response newCustomer(String body) {
        // ...
    }

    @DELETE
    public Response delete(@UriParam("id") int id) {
        // ...
    }
}
```

MessageBodyReader/ MessageBodyWriter

Converts between Java types and representations

Class marked with `@Provider`, implements `MessageBody{Reader|Writer}<T>`

Provides methods for conversion InputStream/OutputStream to/from Java object of type `T`

Example

```java
@Provider
@ProduceMime({"application/vnd.innoq.customer+xml", "text/plain"})
public class CustomerWriter implements MessageBodyWriter<Customer> {
    public boolean isWriteable(Class<?> type) {
        return Customer.class.isAssignableFrom(type);
    }

    public long getSize(Customer customer) {
        return -1;
    }

    public void writeTo(Customer customer, MediaType mediaType,
                           MultivaluedMap<String, Object> httpHeaders,
                         OutputStream entityStream
                       ) throws IOException {
        if (mediaType.getType().compareTo("application") == 0 &&
                mediaType.getSubtype().compareTo("vnd.innoq.customer+xml") == 0) {
            Element root = new Element("customer", NAMESPACE);
            Element name = new Element("name", NAMESPACE);
            name.appendChild(customer.getName());
            root.appendChild(name);
            writeElementToStream(root, entityStream);
        } else if (mediaType.getType().compareTo("text") == 0 &&
                mediaType.getSubtype().compareTo("plain") == 0) {
            OutputStreamWriter osw = new OutputStreamWriter(entityStream);
            osw.write("Customer: \"");
            osw.write(customer.getName());
            osw.write("\"\n");
            osw.close();
        }
    }
}
```

Sub Resource support

Methods annotated with `@Path` without `@GET, @POST, ...` allow for hierarchical resources

Typical use: Collection resources

```
@Path("{id}")
public CustomerResource customerById(@UriParam("id") int id) {
    return new CustomerResource(Customer.get(id));
}
```

Resource hierarchy

/orders

GET - list all orders
PUT - unused
POST - add a new order
DELETE - cancel all orders

/customers

GET - list all customers
PUT - unused
POST - add new customer
DELETE - delete all customers

"Root" resource collections

/orders/{id}

GET - get order details
PUT - update order
POST - add item
DELETE - cancel order

/customers/{id}

GET - get customer details
PUT - update customer
POST - unused
DELETE - delete customer

Sub resources

/customers/{id}/orders

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

Nested resource collection

Response Builder Pattern

Enables creation of objects with additional HTTP metadata

```
return Response
         .status(404)
         .entity("Huh?\n")
         .type("text/plain")
         .build();
```

UriBuilder

Enables creation of URIs without repeating URI template content

Used to support hypermedia - i.e., create links

Builder pattern, again:

```
URI uri = UriBuilder
            .fromUri(BASEURI)
            .path(CustomersResource.class)
            .path(id).build();
```

@HttpContext

`@HttpContext` to access

- URI Info (Class `UriInfo`)
- HTTP Headers (Class `HeaderParam`)
- Preconditions (Class `HttpHeaders`)

Environments

Deployment to multiple different environments:

- Embedded HTTP Server (Java 6)
- Servlets
- Java EE
- JAX-WS
- Others (e.g. Restlet, ...)

Demo

Under Discussion

Several refactorings

Client API

MVC Support (UI Integration)

What you can do

Read the spec!

Download and play with Jersey!

Provide feedback!

Thank you!
Any questions?

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

Architectural Consulting

SOA	WS-*	REST
MDA	MDSD	MDE
J(2)EE	RoR	.NET

innoQ Deutschland GmbH
Halskestraße 17
D-40880 Ratingen
Phone +49 21 02 77 162-100
info@innoq.com · www.innoq.com

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

http://www.innoq.com