

IT Dan 2008
Java conference
Belgrade, Serbia
October 30th, 2008

Have some REST Building Web 2.0 applications and services

Nenad V. Nikolic

Senior Software Engineer, Mogul Balkan
Software Consultant, NVN



Author background check

- Qualifications, experience?
 - Web portals, CMS, applications
 - Graphical apps: GIS, anti-spam
 - networking drivers
 - Integration/middleware
 - Java, C#, C++, Python,
- http://www.xing.com/profile/Nenad_Nikolic
- <http://www.linkedin.com/in/shonzilla>

Web background check

Web 1.0: terminal programming model

Web 1.00001: traces of JavaScript appear

Web 1.1: IE showing JavaScript errors

Web 1.2: web gets the 3rd dimension... alas, buttons only

Web 1.3: built for IE or Netscape

Web 1.4: basic client-side validation, 3D smileys

Web 1.5: Flash says Please wait...

Web 1.6: Popup calendar... not visible in IE4

Web 1.7: *Back* not working - shows porno instead (whoops!)

Web 1.8: Win Server 2003 trusted zone hell

Web 1.9: You play Flash games until Web 2.0 comes

Web 2.0: full-blown email client inside the browser, etc.

What REST is not?

NOT a protocol

NOT a library

NOT HYPE ;-)

What is REST?

REpresentational State Transfer

Architectural style

Roy T. Fielding's **PhD thesis**

(don't worry, **REST** is simple)

Web [2.0] an application domain

What is REST? (cont'd)

Software architecture for building
hypermedia systems
(web is one of them)

Network architecture principles

Way to design data oriented interfaces

We focus on the Web: HTTP + URI

Alternative to traditional web services
RPC, SOA, SOAP - bad to better&complex

REST for the web

REST over HTTP

Sound foundations

Standards with proven track record:

URI - RFC **2396**, 08/1998; de jure

HTTP 1.1 - RFC **2616**, 08/1999; de jure

REST PhD dissertation, 2000; de facto

Standards mash-up -> synergy

Principles of REST

- 1a. Application state -> Resource(s)
- 1b. Functionality -> Resource
- 2. Each resource: uniquely addressable
- 3. Universal addressing scheme
- 4. Uniform interface to xfer representation
- 5. Underlying protocol (HTTP) requirements

Protocol requirements

- ✓ **Client-server**
request-response,
client wants data, server has it
- ✓ **Stateless**
request = independent transaction
- ✓ **Cacheable**
++performance
- ✓ **Layered**
proxies + caches + firewalls



Resource

Source of information
for clients (for caches/proxies too)

Multiple representations
HTML, JSON, XML, image, PDF, ...

Resources hyperlinked
Web things point one to another

REST syntax

Each resource is a noun

Noun - person, account... *anything*

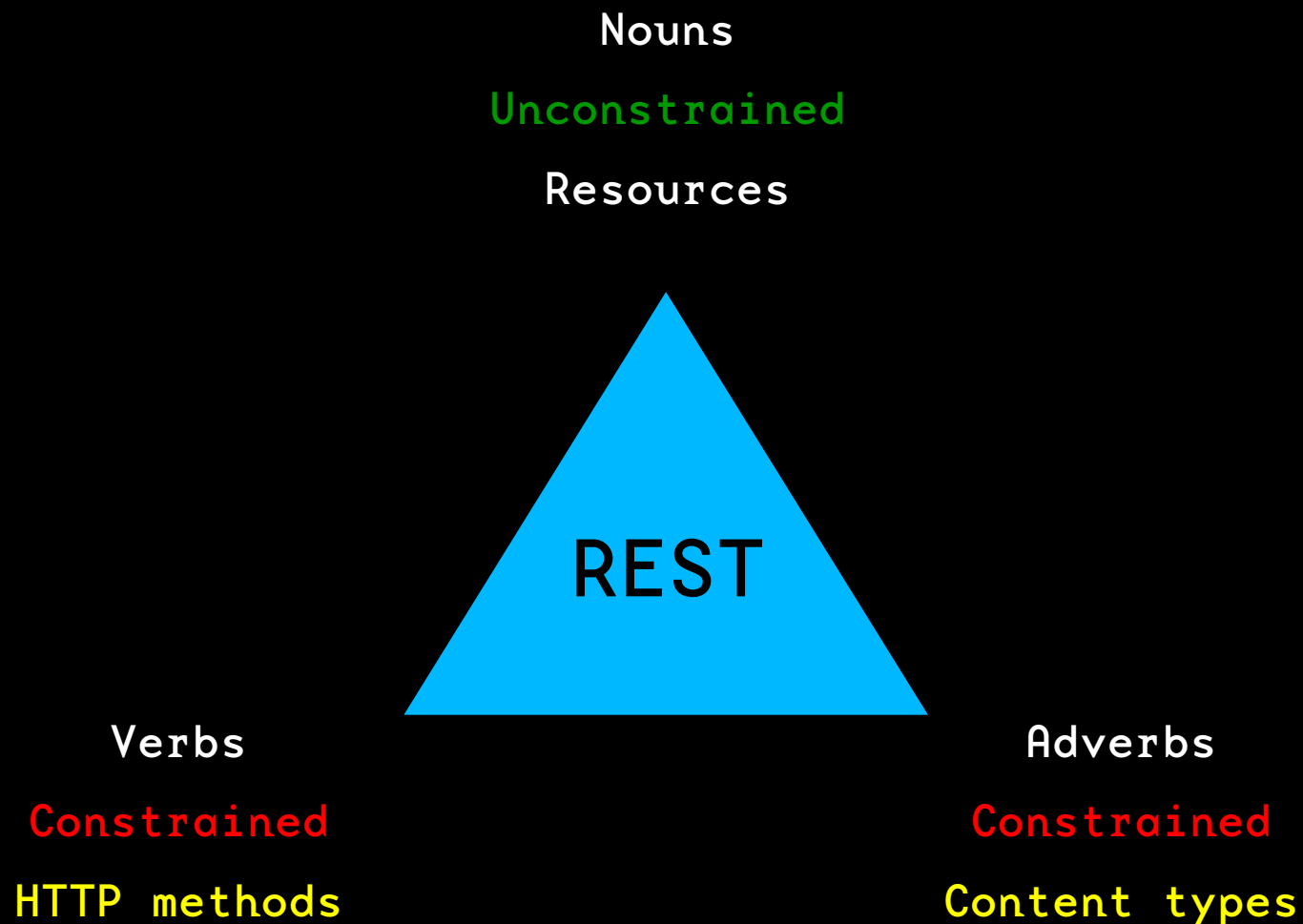
Action over a resource is a verb

Verb - HTTP method

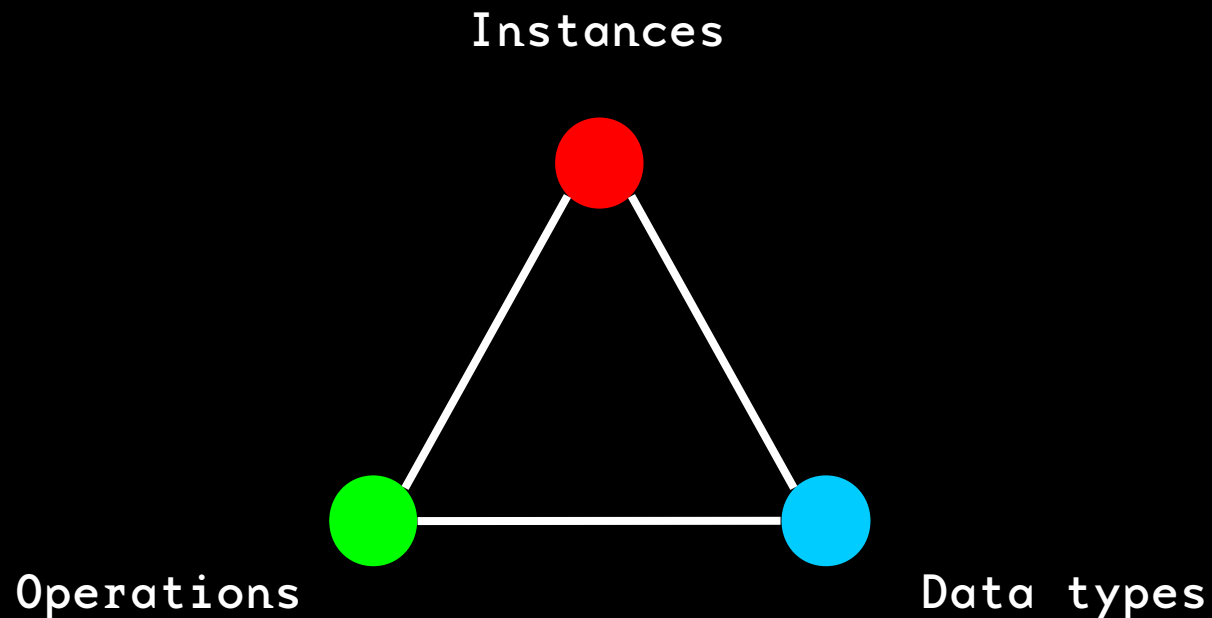
Representation is an adverb

Adverb - content type

REST syntax (cont d)



Techy visualization



Simple, isn't it?

REST principles

- Addressability
- Uniform interface
- Representation oriented
- Stateless communication

Object: addressability

- Resource addresses
- Each address is unique
- Infinite URI naming space
 - <http://shonzilla.com/customers/123>
 - <http://shonzilla.com/customers/politika>
 - <http://shonzilla.com/orders/08/165>
 - <http://shonzilla.com/products/53>
 - <http://shonzilla.com/orders/7865/items/2,5>
 - <http://stanford.edu/students/08/123>
 - <http://stanford.edu/students?name=Sergey&graduated=true>
- Bookmark, biz card, even **tattoo?!?**

Operations: uniform interface

HTTP	GET	POST	PUT	DELETE
Operation	Find	Create	Update	Delete
Database	SELECT	INSERT	UPDATE	DELETE

Data types: representations

- Request metadata - HTTP headers
- ***Content-Type***: HTTP header
- Internet media types aka MIME types
 - *text/html, application/pdf*
 - *application/xml*
 - *application/json*
 - *<type>/<subtype>*
- Content negotiation flexibility

Benefits

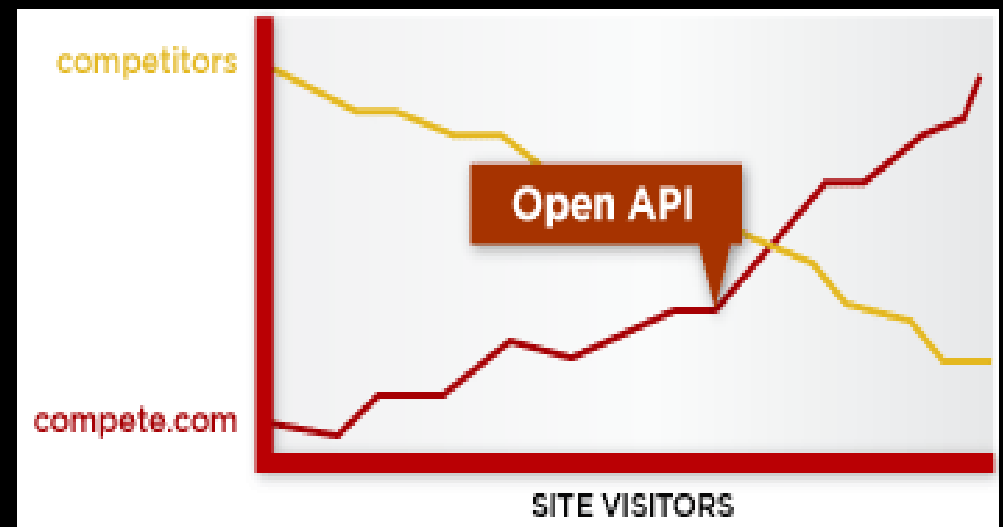
- Ease of use
- Simple and self-descriptive interface
- Decoupling
- Leveraging existing infrastructure
- Caches, proxies

Benefits (cont d)

- Multiple representations
- Accessibility (web clients&servers)
- Same API for servers, web browsers
- ... (many more)
- Web 2.0 minus GUI
- GUI: any AJAX library (incl. **GWT**)

Applications of REST

- Want to develop simple interfaces?
- Have some rest!
- Web services, APIs
- You have *publicly available data*?
You need API!
- Accessible data
can be viral!
- Submitting data can
make it grow faster!
- What is an API?
I call it Biz Dev 2.0!
- Caterina Fake, Flickr



Example: XML representation

- **Client request:**

```
GET http://shonzilla.com/orders/06/81 HTTP/1.1  
Content-Type: application/xml
```

- **Server response:**

```
Content-Type: application/xml  
Content-Length: 202  
<?xml version="1.0"?>  
<order self='http://shonzilla.com/customers/06/81' >  
  <amount currency= RSD >17641.00</amount>  
  <customer ref='http://shonzilla.com/customers/123' />  
  <order-entries>  
    <order-entry>  
      <product ref='http://shonzilla.com/products/53' />  
      <quantity>4</quantity>  
    </order-entry>  
    <order-entry>  
      <product ref='http://shonzilla.com/products/29' />  
      <quantity>2</quantity>  
    </order-entry>  
    <order-entry>  
      <product ref='http://shonzilla.com/products/81' />  
      <quantity>2</quantity>  
    </order-entry>  
  </order-entries>  
</order>
```

Example: PDF representation

- Client request:

GET <http://shonzilla.com/orders/06/81> HTTP/1.1

Content-Type: application/pdf

- Server response:

Content-Length: 53248

Content-Type: application/pdf

NVN D.O. za proizvodnju, inženjering i trgovinu
adresa: Nimodraška 111/17, 11010 BEOGRAD
tel/fax: 011/2471-585 \\\\ tel: 011/2472-785
e-mail: nvn@eurppe.com
PIB: SR100433275, matični broj: 06189911
račun: 180-0214600101000-71

KUPAC:
„Politika“ AD
Makedonska 29
11000 Beograd
PIB: SR100002524

Račun br. 81/06 Datum 23. feb. 2006.

Naziv artikla	Komada	Din/kom.	Ukupni iznos (Din)
1 Ležaj NU2211ECP (SKF)	4	2.870	11.480,00
2 Ležaj 6208-2RS (SNR)	2	840	1.680,00
3 Ležaj 6307-2RS (SNR)	2	895	1.790,00
		PDV 18%	2.691,00
		Ukupno	17.641,00

Roba je isporučena 23. feb. 2006. po Otpremnici br. 81/06.
Rok plaćanja je 30 (trideset) dana.

Prodavac

Example: GET error

- **Client request:**

```
GET http://shonzilla.com/orders/09/999 HTTP/1.1  
Content-Type: application/xml
```

- **Server response:**

```
HTTP/1.1 404 Not Found  
Content-Type: text/html  
Date: Thu, 30 Oct 2008 00:52:09 GMT  
Connection: close  
Content-Length: 26  
<h1>Order not found.</h1>
```

Example: create

- **Client request:**

```
POST http://shonzilla.com/orders HTTP/1.1
```

```
Content-Type: application/xml
```

```
Content-Length: 309
```

```
<?xml version="1.0"?>
```

```
<order self='http://shonzilla.com/customers/08/165' >
```

```
  <amount>23523</amount>
```

```
  <customer ref='http://shonzilla.com/customers/123' />
```

```
  <order-entries>
```

```
    <order-entry>
```

```
      <product ref='http://shonzilla.com/products/53' />
```

```
      <quantity>2</quantity>
```

```
    </order-entry>
```

```
    <order-entry>
```

```
      <product ref='http://shonzilla.com/products/86' />
```

```
    </order-entry>
```

```
  </order-entries>
```

```
</order>
```

Example: created

- **Server response:**

```
HTTP/1.1 201 Created
```

```
Content-Type: text/html
```

```
Date: Thu, 30 Oct 2008 01:18:09 GMT
```

```
Connection: close
```

```
Content-Length: 36
```

```
<h1>Order created successfully.</h1>
```

HTTP on steroids

- HTTP status codes less is more
- Headers instruct the web
 - Representation - **Content-Type**
 - Where next - **Location**
 - Everyone obey this - **Cache-Control**
 - Verify local copy w/ crypto checksum - **ETag**
 - Localization - **Content-Language**
 - Age, Date, Expires, Last-Modified, ...
 - Link: `<mailto:shone@europe.com>; rev= Made"`

RESTful vs. RESTless

- RESTful, RESTlike, RESTless
- HTTP API is NOT REST
- RESTful:
 - Hypertext - resources pointing at each other
 - All HTTP methods, HTTP status codes
 - Obey content types
 - URI design ~ = API design

RESTful vs. RESTless #2

- Overloading POST for create&update
- XHR - some web browsers have issues
 - <http://www.mnot.net/javascript/xmlhttprequest/>
- HTTP method tunneling (XHR, misconfigured nets?)
 - **Anti-pattern**: operation in URI, etc.
 - Header (**GData**): **X-HTTP-Method-Override**
 - Query parameter (Ruby on Rails tunnels thru POST):
GET <http://shonzilla.com/orders/13&method=delete> HTTP/1.1
POST <http://shonzilla.com/orders/&method=delete> HTTP/1.1

REST API by the REST man

- **URI is enough.** Everything else should be figured out by the client.
- *A REST API should be entered with no prior knowledge beyond the initial URI (bookmark) and set of standardized media types that are appropriate for the intended audience (i.e., expected to be understood by any client that might use the API). From that point on, all application state transitions must be driven by client selection of server-provided choices that are present in the received representations or implied by the user's manipulation of those representations. The transitions may be determined (or limited by) the client's knowledge of media types and resource communication mechanisms, both of which may be improved on-the-fly (e.g., code-on-demand).*
- Roy T. Fielding, REST APIs must be hypertext-driven
- **This requires quite a lot from a client.**
- **Such a generic REST client would be universal**

REST vs. RPC & SOAP

- SOAP, WSDL, UDDI
- RPC tight coupling
- WS-* over-designed; WS Death Star
- Insufficient tooling (until soapUI)
- XML schemata obsolete SOAP
- REST: flexibility + faster processing

Java API for REST

- JAX-RS a.k.a. **JSR 311**, now *final*
- Will be included in Java EE 6 (**JSR 316**)
- Java API for RESTful web services
- Closely related alphabet soup:
 - XML, JSON (JavaScript Object Notation)
 - **JAXB** 2 (Java Architecture for XML Binding)
 - RSS (Really Simple Syndication)
 - **APP** (Atom Publishing Protocol)
- URI mapping using UriTemplate
- Resource life-cycles: 1/request, 1/application

REST libs & frameworks

- **Jersey**, reference implementation
- **Restlet**, lightweight framework
- Apache CXF, implements WS-* too
- JBoss **RESTEasy**
- NetKernel (RESTful app server)
- Axis2, don't even bother
- Spring 3.0 (in Jan 2009)
- ...

Java code, finally

- Use JAX-RS: `import javax.ws.rs.*`
- Define a resource class
- Pass parameters to resource
- Implement GET method(s)
- MIME typed custom representation
- Implement POST method
- Support multiple representations w/ JAXB
- Jersey life-cycle support: singleton and session resources

Resource class that GETs

```
// JAX-RS resource skeleton
@Path("/orders")
public class OrderResource {
    @GET public String getOrders() { }
}
```

Resource class that GETs...

```
// JAX-RS resource skeleton
@Path("/orders")
public class OrderResource {
    @GET public String getOrders() { }

    // GET /orders/523
    @GET @Path("/{id}")
    public String getOrder(@PathParam("id") int orderId) {
        // get order object,
        // extract attributes, build response
    }
}
```

Resource class that GETs it

```
// JAX-RS resource skeleton
@Path( /orders")
public class OrderResource {
    // ...
    @GET @Path("/{id}")
    @Produces("text/xml")
    public String getOrder(@PathParam("id") int orderId) {
        // get order object,
        // extract attributes, prepare response
        return response;
    }
}
```

Resource class that searches

```
// JAX-RS resource skeleton
@Path("/orders")
public class OrderResource {
    // ...
    @GET @Path("/{search}")
    @Produces("text/xml")
    public String getOrder(@PathParam("search") String query) {
        // search searchable order attributes for query text,
        // extract attributes for each result, prepare response
        return response;
    }
}
```

Support representations

- Method overriding problem?
- No problem
- Resource method names - anything
- How about repeating code?
- Hold on... we handle this elegantly

Resource class that GETs it

```
// JAX-RS resource skeleton
@Path("/orders")
public class OrderResource {
    // ...

    @GET @Path("/{id}") @Produces("text/xml")
    public String getOrder(@PathParam("id") int orderId) { }

    @GET @Path("/{id}") @Produces("text/html")
    public String getOrderHtml(@PathParam("id") int orderId) { }

    @GET @Path("/{id}") @Produces("application/json")
    public String getOrderJson(@PathParam("id") int orderId) { }
}
```

Custom representation using a MIME type

- Building different representations
- Again and again? No, thanks!
- `java.ws.rs.ext.*` to the rescue
- response body builder (marshaller)
- Built-in support for XML and JSON using JAXB
- JAXB annotations are needed

Annotated resource entity classes

```
@XmlElement(name="order")
public class ApiOrder {
    @XmlAttribute(name="id") int id;
    @XmlElement(name="customer-id")
    int customerId;
    @XmlElement("order-entries")
    List<ApiOrderEntry> entries;
    @XmlElement( amount")
    ApiNumber amount;
    // ...
}
```

Return any representation

```
// JAX-RS resource skeleton
@Path("/orders")
public class OrderResource {
    // ...
    // Marshaller will take care
    @GET @Path("/{id}")
    @Produces({"application/xml",
              "application/json"})
    public ApiOrder getOrder(@PathParam("id") int
                             orderId) { }
}
```

Resource class that POSTs

```
// JAX-RS resource skeleton
@Path("/orders")
public class OrderResource {
    // ...
    // POST /orders
    // (with order XML data in request body)
    @POST @Consumes("application/xml") public
    Response createOrder(ApiOrder newOrder) { }
}
```

Updating an order with PUT

```
// JAX-RS resource skeleton
@Path("/orders")
public class OrderResource {
    // ...
    // PUT /orders/512
    // (w/ data in request body)
    @PUT @Path("/{id}")
    @Consumes("application/xml")
    public Response createOrder(
        @PathParam("id") int orderId,
        ApiOrder newOrder) { }
}
```

HTTP status codes

```
// JAX-RS resource skeleton

@Path("/orders")

public class OrderResource {

    // ...

    @GET @Path("/{id}") @Produces({ "application/xml", "application/json"})
    public Response getOrder(@PathParam("id") int orderId) {
        Order order = OrderService.getOrder(orderId);
        ApiOrder apiOrder = ApiOrderFactory.getOrder(order);
        Response.ResponseBuilder response = Response.ok(apiOrder);
        Date now = new Date();
        Date paymentDeadline = OrderService.calcPaymentDeadline(now);
        resp.expires(paymentDeadline);
        return response;
    }
}
```

Error handling

```
// JAX-RS resource skeleton

@Path("/orders")

public class OrderResource {

    // ...

    @GET @Path("/{id}")
    @Produces({ "application/xml", "application/json"})
    public ApiOrder getOrder(@PathParam("id") int orderId) {
        Order order = OrderService.getOrder(orderId);
        if (order == null) {
            throw new NotFoundException( order + orderId);
        }
        ApiOrder apiOrder = ApiOrderFactory.getOrder(order);
        return apiOrder;
    }
}
```

Custom exceptions

```
public class NotFoundException extends
    WebApplicationException {
    public NotFoundException() {
        super(Response.Status.GONE);
    }
    public NotFoundException(String msg) {
        super(Response.Status.GONE, msg);
    }
}

// WebApplicationException is a run-time exc.
```

More JAX-RS annotations

- **@Singleton** - singleton resource instance
- **@QueryParam**, **@HeaderParam**, **@MatrixParam**
- **@DefaultValue** - default param value (if one is omitted from URI)
- **@Context** - inject [UriInfo, Request, HttpHeaders, SecurityContext] into a class field, property or method param
- **@CacheControl** - resource controls cache
- **@Cookie**, **@CookieParam** - NOTE: propagating server-side session state is an **anti-pattern**

Other topics

- AAA = authentication + authorization + accounting (per resource|method)
- Accounting = request logging
- Request throttling (req/min, etc.) - to reduce server load
- *In-container API unit testing*
- *API/URI design and versioning*
- **application.wadl** file shows the available resources menu
- RIA using REST:
 - GWT (Google Web Toolkit)
 - Adobe AIR and Flex
 - JavaFX
- Stay tuned...

REST protocols

- REST application protocols
- Interacting with the web
 - Publish and edit Web resources
- APP (Atom Publishing Protocol), RFC 5023
 - Implemented under Apache Abdera project
- GData (Google data APIs)
 - APP + queries + auth + optimist.lock.
 - Contacts, Calendar, Docs, ...

REST Clients

- web browsers (Web 2.0 clients)
- Web services clients
- Integration systems (Mule, ServiceMix)
- Mobile devices (Google Android)
- Web apps (Google Services: Calendar, Docs, Contacts, etc.)

Famous APIs

- Web API == HTTP API != REST API
- Flickr - RESTless: delete is POSTed
- Amazon S3 - RESTlike and SOAPy storage
- Digg - social news, read-only and clean API
- Twitter - RESTful microblogging
- Yelp - RESTful yellow pages
- Delicious - RESTless social bookmarking
- New York Times (!)
- and many more (see Programmable web)

Beyond API API mashups

- DIY - learn REST, JAX-RS, etc.
- Commercial outsourcing:
 - **Mashery** (on-demand API infrastructure)
 - **email me** (custom API R&D)
- **GNIP** - data portability & filtering
- More to come...
- Stay tuned: **Shonzilla**, **JavaSvet**

GET /applause HTTP/1.1

200 OK Thank you!

http://shonzilla.com/docs/it-dan-2008_have-some-rest.pdf

