REST For Web Services
Course, Fall 2005

Henning Niss

IT University of Copenhagen

Lecture 7
Material

- **Main material (curriculum / “pensum”):**

- **Supplementary material:**
  - **REST Wiki:** [http://rest.blueoxen.net/cgi-bin/wiki.pl](http://rest.blueoxen.net/cgi-bin/wiki.pl)
    [http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html](http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html)
Overview

Example: RESTful Web services

The REST architectural style

Example: REST in the Joke server

REST Overview

Example: RESTful Web services

The REST architectural style

Example: REST in the Joke server
Introduction

REST is an architectural style.

- “packaged set of architectural design decisions”,
- a software architecture abstracts runtime elements of a system.

Goals:

- idealized model of interactions within a Web application,
- function as guiding principles to avoid flaws,
- minimize network latency and communication,
- maximize independence and scalability.

Most prominent example: the modern Web.

Focus today: REST from a Web services perspective.
REST-based Web services

Very high-level description of REST-based Web services:

resource-centric XML-over-HTTP services.

- endpoints are resources;
- endpoints are accessible via standard HTTP;
- endpoints are represented in XML.
Applying REST — example

The successful company Parts Depot, Inc sells many different parts to many customers.

They would like well-designed Web services for this.

Parts Depot Web services:

- get list of parts;
- get detailed part information;
- submit purchase orders (PO).

Example due to R.L. Costello: *Building Web Services the REST Way*. 
Getting a part list

The list of parts available from Parts Depot is a resource:

- a resource is *identified* via a URI;

- resource representations are available via *standard HTTP GET*:
  
  ```
  GET http://www.parts-depot.com/parts HTTP/1.1
  ```

- representations may be *adapted* to consumer:

  ```
  <?xml version="1.0"?>
  <p:Parts xmlns:p="http://www.parts-depot.com"
    xmlns:xl="http://www.w3.org/1999/xlink">
    <Part id="0345" xl:href="http://www.parts-depot.com/parts/0345"/>
    <Part id="0346" xl:href="http://www.parts-depot.com/parts/0346"/>
    <Part id="0347" xl:href="http://www.parts-depot.com/parts/0347"/>
    <Part id="0348" xl:href="http://www.parts-depot.com/parts/0348"/>
  </p:Parts>
  ```

- a representation typically *contains links* to other resources.
Getting part information

Detailed part information is also a resource identified with a URL.

▶ getting detailed information:
GET http://www.parts-depot.com/parts/0346 HTTP/1.1

▶ representation:
<?xml version="1.0"?>
    xmlns:xl="http://www.w3.org/1999/xlink">
    <Part-ID>0346</Part-ID>
    <Name>Widget-A</Name>
    <Desc>This part is used within the frap assembly</Desc>
    <UnitCost currency="USD">0.10</UnitCost>
</p:Part>

▶ *how* representation is generated is unspecified;
▶ *logical* URI (not necessarily static pages).
Submitting a purchase order

To order a part, the client submits a purchase order.

- an order submitted using *standard HTTP POST*;
- the order is an *XML document*

```xml
<?xml version="1.0"?>
<p:Order xmlns:p="http://www.parts-depot.com"
         xmlns:xl="http://www.w3.org/1999/xlink">
  <Quantity>10</Quantity>
  <Date>2005-10-11</Date>
  ...
</p:Order>
```

- an order is *shared information* between Parts Depot and client;
- the Web service *returns a URI* for the submitted order;
- the client may delete the order using *HTTP DELETE*.
Assessment (example)

Salient points of the example:

- focus on resources (resource-centric vs API-centric);
- resources are identified using URIs (“name everything”);
- resources are connected through links (“reveal gradually”);
- resources may have different representations (HTML, XML, ...);
- based on standard technology (URI, HTTP, XML, ...);
- separation of updatable and immutable resources (POST & GET).
The name REST

The name “Representational State Transfer” is intended to evoke an image of how a well-designed Web application behaves: a network of Web pages forms a virtual state machine, allowing a user to progress through the application by selecting a link or submitting a short data-entry form, with each action resulting in a transition to the next state of the application by transferring a representation of that state to the user.

(Fielding & Taylor, 2002)
The major goal of the Web was to be a shared information space for people and machines to communicate.

- Low entry-barrier.
- Extensibility.
- Distributed hypermedia.
- Internet scale (scalable & independent).

REST is an architectural style for “architecting the Web”:

- “capture” the success of the Web;
- guide future extensions.
REST Constraints (1)

REST consists of a number of architectural constraints:

- **client/server constraint:**
  - separation of concerns
  - independent evolution of components;

- **stateless constraint:**
  - visibility (single request reveals everything)
  - reliability (easier to recover from failures)
  - scalability (server does not commit resources to each request)
  - degrades network performance (by requiring large requests);

- **caching constraint:**
  - improves network efficiency (eliminates interactions)
  - clients have the “right” to reuse cacheable data
  - degrades reliability (due to stale data);
REST Constraints (2)

- uniform interface constrain:
  - 😊 general component interface
  - 😊 implementations decoupled from interfaces
  - 😣 degrades efficiency (to adapt to the uniform interface);

- layered system constraint:
  - 😊 architecture can be composed hierarchically
  - 😊 overall system complexity bounded
  - 😣 added overhead;

- (code-on-demand)
Architectural elements

Three classes of architectural elements:
- processing elements (components);
- data elements (resources);
- connecting elements (connectors).

REST is concerned with architecture; for example
- component implementation left to service providers;
- protocol syntax left to HTTP and XML.
Resources

A resource is “anything that can be named”.

▶ a document,
▶ “today’s weather in Los Angeles”,
▶ a collection of other resources.

Formally: membership function mapping a time \( t \) to a set of values (“representations”).

▶ the empty set (resource not yet realized),
▶ an XML representation and a HTML representation,
▶ a(nother) resource identifier (“redirect”).

Resources identified or named by resource identifiers (URIs).

The identifier (not the content) captures the “semantics” of the resource.
Representations

A representation captures the current state of a resource.

A representation is computed ("rendered") when a resource is requested ("late binding").

Where?

- process data where it is located and send result to client,
- encapsulate data in an object and send object to client,
- send data and metadata so client can determine processing.

All may be employed by REST.

A representation is both data and metadata.

- representation data typically in XML;
- metadata typically in HTTP headers.
Connectors

A **connector** is an abstract interface for network communication.

Key requirement: **stateless**.
- each request includes all necessary information to process it,
- no resources need be held,
- need no knowledge of interaction semantics with other requests,
- intermediaries may inspect communication in isolation.

Interface:
- in params: control data, resource id, optional resource repr,
- out params: control data, optional resource/repr metadata, optional resource repr.

*Synonymous *invocation*, but may use streams for input and output.

Example connectors: client, server, cache.
Processing elements (components) “performs” requests and responses.

Example processing elements: user agent and origin server.

- A user agent (uses a client connector)
  - sends requests and receives replies;
  - renders the result;
  - example: Web browser.

- An origin server (uses a server connector)
  - serves requested resources;
  - resource implementation details not revealed;
  - must be ultimate endpoint for updates.

Example: Proxies (acting as both clients and servers).
Architectural views

Views describe how the architectural elements works together.

- A process view shows how data flows through the system.
- A connector view shows how components communicate.
- A data view shows how the state of the system evolves.
Example view (process view)
Web of Jokes

Infrastructure for a world-wide network of servers and clients for authoring, publishing, and reading jokes.
Example: REST in the Joke server

**Operations**

Joke server operations:

- list (joke categories);
- retrieve(category) (jokes of a specified category);
- submit() (joke in body of POST request).
Example: REST in the Joke server

Joke server (List)

```java
public class List extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws IOException, ServletException
    {
        try {
            resp.setContentType("text/xml");
            Document doc = new Jokes(getServletContext()).getJokes();
            String xslt =
                    getServletContext().getInitParameter("CategoriesXSLT");
            XSLTransformer t = new XSLTransformer(xslt);
            new XMLOutputter().output(t.transform(doc), resp.getWriter());
        } catch (Exception e) {
            resp.sendError(500, "Internal error");
        }
    }
}
```
Example: REST in the Joke server

Joke server (List stylesheet)

```xml
<xsl:stylesheet version="1.0"
    xmlns="http://www.brics.dk/ixwt/categories"
    xmlns:jml="http://www.brics.dk/ixwt/jokes"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="jml:collection">
    <categories>
      <xsl:apply-templates select="jml:joke[not(@category=following::jml:joke/@category)]"/>
    </categories>
  </xsl:template>
  <xsl:template match="jml:joke">
    <category>
      <xsl:value-of select="@category"/>
    </category>
  </xsl:template>
</xsl:stylesheet>
```
public class Retrieve extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {
        try {
            resp.setContentType("text/xml");
            Document doc = new Jokes(getServletContext()).getJokes();
            Transformer t =
                TransformerFactory.newInstance()
                .newTransformer(new StreamSource(new File(
                    getServletContext().getInitParameter("RetrieveXSLT"))));
            t.setParameter("category", req.getParameter("category"));
            t.transform(new JDOMSource(doc), new JDOMResult());
            new XMLOutputter().output(out.getDocument(), resp.getWriter());
        } catch (Exception e) {
            resp.sendError(500, "Internal error");
        }
    }
}
public class Submit extends HttpServlet {
    static Namespace jml =
            Namespace.getNamespace("http://www.brics.dk/ixwt/jokes");
    public void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws IOException, ServletException
    {
        PrintWriter out = resp.getWriter();
        SAXBuilder b = new SAXBuilder();
        // set schema validation properties ...
        try {
            Document doc = b.build(req.getInputStream());
            if (!doc.getRootElement().getNamespaceURI().equals(jml))
                resp.sendError(400, "Wrong namespace of root element!");
            else
                new Jokes(getServletContext()).addJokes(doc);
        } catch (Exception e) {
            resp.sendError(500, "Internal error");
        }
    }
}
Jokes database (I)

```java
public class Jokes {
    static Namespace jml =
        Namespace.getNamespace("http://www.brics.dk/ixwt/jokes");
    ServletContext context;
    String jokeFile;
    Document jokes;

    public Jokes(ServletContext ctx) throws JDOMException, IOException {
        context = ctx;
        jokes = (Document)ctx.getAttribute("jokes");
        jokeFile = ctx.getInitParameter("JokeFile");
        if (jokes==null) {
            try {
                jokes = new SAXBuilder().build(new File(jokeFile));
            } catch (Exception e) {
                jokes = new Document(new Element("collection", jml));
            }
        }
        ctx.setAttribute("jokes", jokes);
    }
}
```

Henning Niss (ITU)
Jokes database (II)

```java
public Document getJokes() { return jokes; }

public void addJokes(Document m) throws JDOMException, IOException {
    synchronized(context) {
        jokes.getRootElement()
            .addContent(m.getRootElement().removeContent());
        new XMLOutputter()
            .output(jokes, new FileOutputStream(new File(jokeFile)));
    }
}
```
Summary

- resource-centric;
- resources are named (identified);
- representations rather than resources;
- based on standardized protocol (HTTP);
- focus on architecture rather than implementation detail.