REST web services

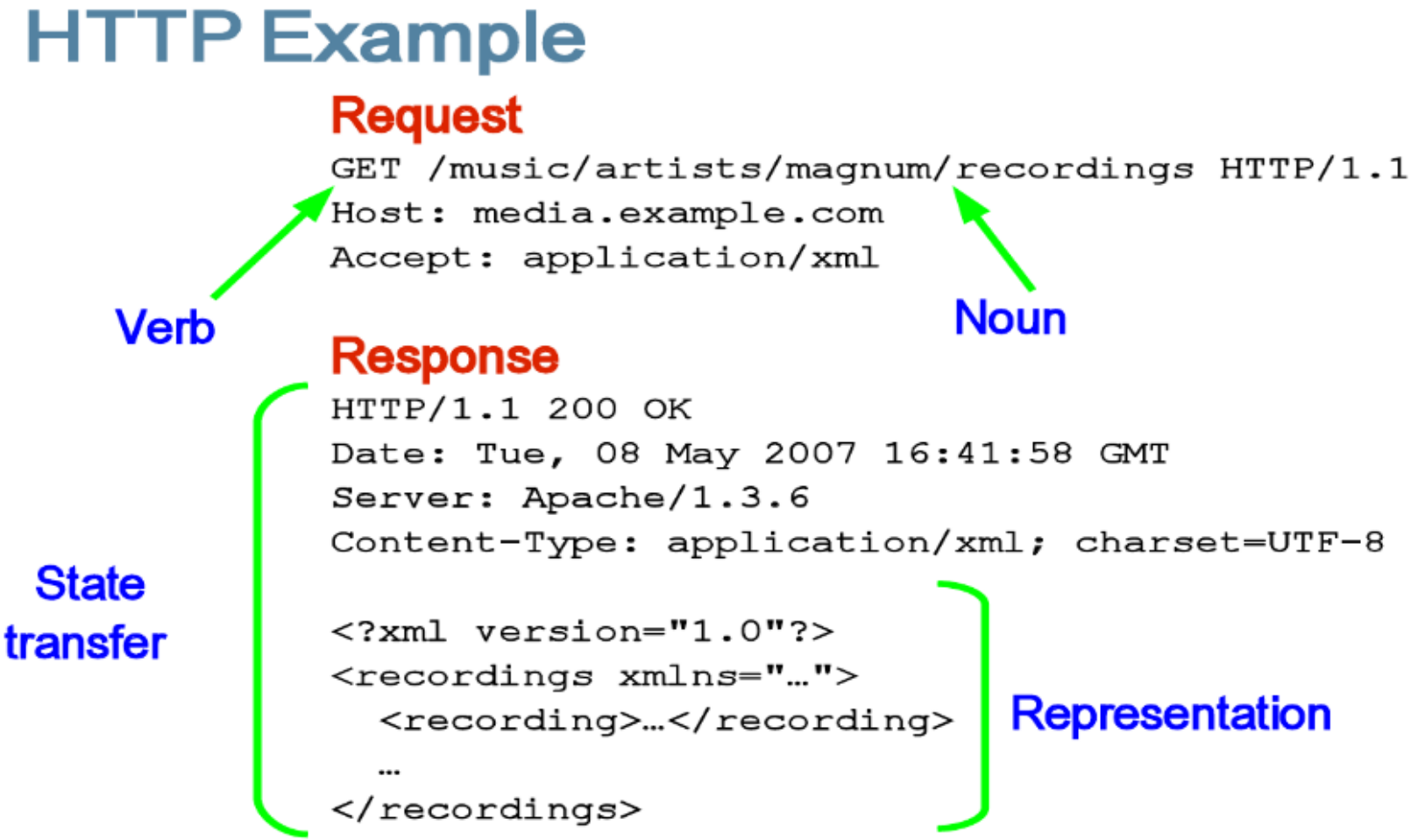
Representational State Transfer

Author: Nemanja Kojic

What is REST?

- Representational State Transfer (ReST)
- Relies on stateless, client-server, cacheable communication protocol
- It is NOT a standard
- It is an architecture style for designing networked applications
- In almost all cases HTTP protocol is used (instead of complex CORBA, RPC or SOAP)
- HTTP can be viewed as REST-based architecture, as well
- HTTP used to post, read and delete data

What is REST? (cont.)



REST as Lightweight Web Services

- It is a programming approach
- REST is a lightweight alternative to complex mechanisms like:
  - RPC (Remote Procedure Call)
  - Web services (SOAP, WSDL,…)
- REST service is:
  - Platform independent
  - Langugage independent
  - Standards based (runs on top of HTTP)
  - Can easily be used in presence of firewalls.

REST as Lightweight Web Services

- REST offers no:
  - Built-in security features
  - Encryption
  - Session management
  - QoS quaranties
- Security features can be added easily:
  - Security: user/pass tokens
  - Encryption: HTTPS (secure sockets)

REST as Lightweight Web Services

- Cookies is not part of good REST design
- REST operations are self-contained
- Each request carries with it (Transfers) all the information (State) that the server needs to complete it

REST vs. SOAP request

SOAP Request

HTTP Post

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

SOAP message must be assembled properly, and it is included as the HTTP payload.

The result may be an XML content embodied inside a SOAP response envelope.

REST Request (URL)

HTTP GET

```
http://www.acme.com/phonebook/UserDetails/12345
```

HTTP reply is the raw result data, as-is.

URL's method is called UserDetails, instead of GetUserDetails. It is a common convention in REST design to use nouns rather than verbs to denote simple resources.

REST vs. SOAP

- SOAP-based web Services are often implemented with libraries that maintain SOAP/HTTP requests
  - Create and send the SOAP request
  - Parse the SOAP response
- With REST, only a simple network connection is used
- Still there are some useful libraries that simplify the REST things.

More complex REST requests

- REST can easily handle more complex requests (including multiple parameters)
- For passing long parameters,
  or even binary ones,
  one can use HTTP POST requests

```
http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe
```

The REST rules

- GET requests for read-only queries (SHOULD NOT change the state)
- POST requests for Create/Update/Delete
- POST can also be used for read-only queries (large parameters)

REST and XML

- REST services may use XML in their responses
- REST requests rarely use XML
  - request parameters are simple (no need for XML structuring)
- XML response SHOULD BE verified

REST Response Format

- Response is often an XML file
  - XML is easy to expand
- REST is not bound only to XML
- REST can also use other formats:
  - CSV (Comma Separated Values) more compact
  - JSON (JavaScript Object Notation) easily parseable by JavaScript clients
- HTML is not acceptable for REST responses! (except in rare cases, www)

JSON Example

```
{ "empinfo" :
   {
        "employees" : [
        {
          "name" : "Scott Philip",
          "salary" : £44k,
          "age" : 27,

        },

        {
          "name" : "Tim Henn",
          "salary" : £40k,
          "age" : 27,
        },

        {
         "name" : "Long Yong",
         "salary" : £40k,
         "age" : 28,

        }
                  ]
   }
}
```

REST Response Example

```
<parts-list>
 <part id="3322">
  <name>ACME Boomerang</name>
  <desc>
   Used by Coyote in <i>Zoom at the Top</i>, 1962
  </desc>
  <price currency="usd" quantity="1">17.32</price>
  <uri>http://www.acme.com/parts/3322</uri>
 </part>
 <part id="783">
  <name>ACME Dehydrated Boulders</name>
  <desc>
   Used by Coyote in <i>Scrambled Aches</i>, 1957
  </desc>
  <price currency="usd" quantity="pack">19.95</price>
  <uri>http://www.acme.com/parts/783</uri>
 </part>
</parts-list>
```

Real REST Examples

- The Google Glass API (“Mirror API”)
- Twitter REST API https://dev.twitter.com/docs/api
- Flickr, https://www.flickr.com/services/api/
- Amazon, Simple Storage Service http://docs.aws.amazon.com/AmazonS3/2006-03-01/API/APIRest.html
- Atom, restfull variant of RSS
- Tesla Model S (car systems ⇔ Android/iOS apps), http://docs.timdorr.apiary.io/

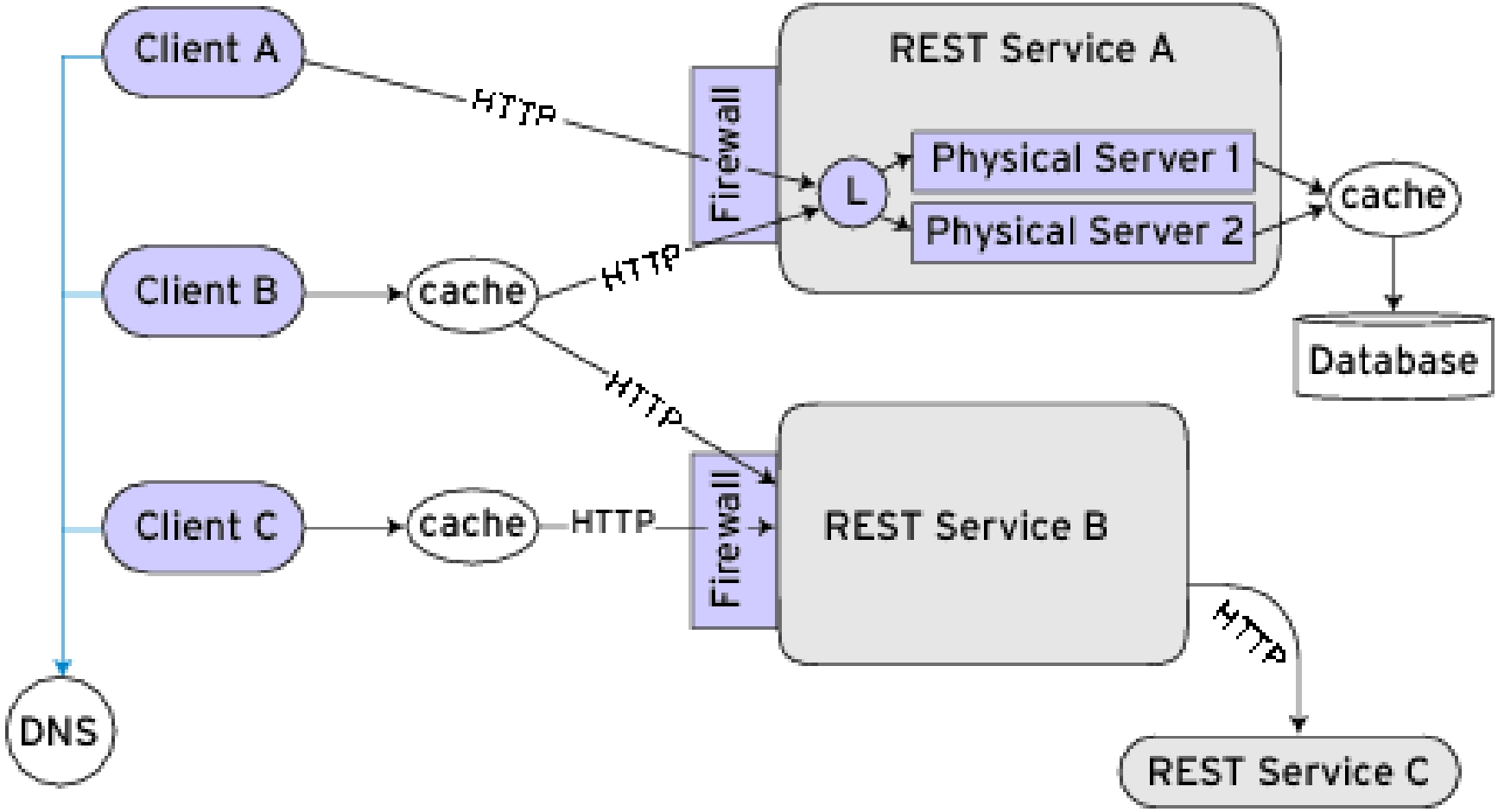
AJAX and REST

- AJAX – Asynchronous JAvaScript and XML
- AJAX makes web pages interactive using JS
- AJAX requests sent as XmlHttpRequest objects
- AJAX response is parsed by JS code
- AJAX follows the REST principles:
  - XmlHttpRequest can be viewed as a GET request
  - Response is often JSON (popular for REST)

REST Architecture Components

- **Resources**
  - Identified by logical URLs
  - Both state and functionality exposed as resources
- **A web of resources**
  - Resources SHOULD NOT be too large and contain too fine-grained data
  - Resources should exploit links to additional data
- **Client-Server**
  - in a distributed fashion
- **No connection state**
  - Stateless communication, although server and client can be stateful.
- **Cacheable resources**
  - Protocol (HTTP) must allow caching of resources (with expiration)
- **Proxy servers**

REST Architecture



REST Design Guidlines

- Don’t use physical URLs.
  - NO: http://www.acme.com/p/product003.xml
  - YES: http://www.acme.com/p/product/003
- Queries should not return an overload data.
  - Provide a paging mechanism with prev/next links
- Make sure the rest response format is well documented
  - In case of XML, provide a schema or DTD

REST Design Guidlines

- Don't let clients construct REST URLs.
  - Provide clients with complete URLs, instead.
  - Include URL with each item.
  - NO: http://www.acme.com/product/*PRODUCT_ID*
  - YES: http://www.acme.com/product/001263
- GET requests should never case a state change.
  - Server state should be changed through POST requests

Documenting REST Services: WADL

- Web Application Description Language
- Created by Sun Microsystems
- WADL is lightweight
- Easy to understand and write
- It is not as flexible as WSDL
- WSDL can also be used for documenting REST services

WADL vs. WSDL

- Often used to describe SOAP-based services
- Flexible in service binding options
  - It is possible to send SOAP messages via SMTP!
- It is more complex than WADL
- Supports all HTTP verbs
- It is acceptable for documenting REST services

WADL: An example

```
<method name="GET" id="ItemSearch">
 <request>
  <param name="Service" style="query"
   fixed="AWSECommerceService"/>
  <param name="Version" style="query" fixed="2005-07-26"/>
  <param name="Operation" style="query" fixed="ItemSearch"/>
  <param name="SubscriptionId" style="query"
   type="xsd:string" required="true"/>
  <param name="SearchIndex" style="query"
   type="aws:SearchIndexType" required="true">
    <option value="Books"/>
    <option value="DVD"/>
    <option value="Music"/>
  </param>
  <param name="Keywords" style="query"
   type="aws:KeywordList" required="true"/>
  <param name="ResponseGroup" style="query"
   type="aws:ResponseGroupType" repeating="true">
    <option value="Small"/>
    <option value="Medium"/>
    <option value="Large"/>
    <option value="Images"/>
  </param>
 </request>
 <response>
  <representation mediaType="text/xml"
   element="aws:ItemSearchResponse"/>
 </response>
</method>
```

The description of Amazon's ItemSeach service

The Complete WADL spec. for Amazon's web service http://www.w3.org/Submission/wadl/#x3-35000A.1

WADL: XML Header Section

```
<application xmlns="http://wadl.dev.java.net/2009/02"
  xmlns:aws="http://webservices.amazon.com/AWSECommerceService/2005-07-26"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd">

  <grammars>
    <include href="AWSECommerceService.xsd"/>
  </grammars>

  <resources base="http://webservices.amazon.com/onca/">
    <resource path="xml">
      <method href="#ItemSearch"/>
    </resource>
  </resources>
```

Namespace declarations.

The XML validation schema.

Using REST in C#

- Issuing HTTP GET request
  - Key classes: HttpWebRequest, HttpWebResponse
  - URL parameters must be properly encoded (%20), use System.Web.HttpUtility.UrlEncode()

```
static string HttpGet(string url) {
  HttpWebRequest req = WebRequest.Create(url) as HttpWebRequest;
  string result = null;
  using (HttpWebResponse resp = req.GetResponse() as HttpWebResponse)
  {
    StreamReader reader = new StreamReader(resp.GetResponseStream());
    result = reader.ReadToEnd();
  }
  return result;
}
```

Using REST in C# (cont.)

- Issuing HTTP POST requests

```
static string HttpPost(string url, string[] paramName, string[] paramVal) {
  HttpWebRequest req = WebRequest.Create(new Uri(url)) as HttpWebRequest;
  req.Method = "POST";
  req.ContentType = "application/x-www-form-urlencoded";
  // Build a string with all the params, properly encoded.
  StringBuilder paramz = new StringBuilder();
  for (int i = 0; i < paramName.Length; i++) {
    paramz.Append(paramName[i]).Append("=").Append(HttpUtility.UrlEncode(paramVal[i])).Append("&");
  }
  // Encode the parameters as form data:
  byte[] formData = UTF8Encoding.UTF8.GetBytes(paramz.ToString());
  req.ContentLength = formData.Length;
  // Send the request:
  using (Stream post = req.GetRequestStream()) {
    post.Write(formData, 0, formData.Length);
  }
  // Pick up the response:
  string result = null;
  using (HttpWebResponse resp = req.GetResponse() as HttpWebResponse) {
    StreamReader reader = new StreamReader(resp.GetResponseStream());
    result = reader.ReadToEnd();
  }
  return result;
}
```

References

- Learn Rest: A tutorial, http://rest.elkstein.org/