Chapter VI –

RESTful Web Services For Value Added Services (VAS) in NGNs

http://users.encs.concordia.ca/~glitho/
Telecommunication Services Engineering (TSE) Lab

Outline

1. Introduction to Web Services
2. RESTful Web Services Overview
3. A Case Study
Introduction to Web Services

1. Definition and principles
2. Overall business model
3. Technologies
Web Services so far

- RESTful Web Services
  - Most `recent``

- SOAP – BASED WEB SERVICES
  - Sometimes called `Big`` Web Services

- This part of the course will discuss the common characteristics
Definitions and principles

Today

• Publication of documents
• Human interaction
• Proprietary ad-hoc interfaces

Tomorrow

• Publication of “reusable business logic”
• Automated Program to program interaction
• Industry standard interfaces

Note: There are other technologies such as JSON that may be used
Definitions and principles

“The term Web Services refers to an architecture that allows applications (on the Web) to talk to each other. Period. End of statement”

Adam Bobsworth in ACM Queue, Vol1, No1
Definitions and principles

The three fundamental principles, still according to Adam Bobsworth:

1. Coarse grained approach (i.e. high level interface)
2. Loose coupling (e.g. application A which talks to application B should not necessarily be re-written if application B is modified)
3. Synchronous mode of communication, but also asynchronous mode
Definitions and principles

- Web 1.0, or the human web, is designed for human use.

- Web 2.0, or the programmable web, is designed for consumption by software programs.

- Web 2.0 enables communities and web client participation.
## Definitions and principles

<table>
<thead>
<tr>
<th>Web 1.0</th>
<th>Web 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>– Human web</td>
<td>– Programmable web</td>
</tr>
<tr>
<td>– Is about HTML</td>
<td>– Is about XML</td>
</tr>
<tr>
<td>– Is about client-server</td>
<td>– Is about peer-to-peer</td>
</tr>
<tr>
<td>– Is about reading</td>
<td>– about writing</td>
</tr>
<tr>
<td>– Is about companies</td>
<td>– Is about communities</td>
</tr>
<tr>
<td>– Is about home pages</td>
<td>– Is about blogs</td>
</tr>
<tr>
<td>– Is about owning</td>
<td>– Is about sharing</td>
</tr>
<tr>
<td>– Is about services sold over the web</td>
<td>– Is about web services</td>
</tr>
</tbody>
</table>

...
Business model

Broker (Human + agent)

Requestor (Human + agent)

Provider (Human + agent)
Business model

Requestor
• Person or organization that wishes to make use of a Web service.
• Uses an agent (i.e requestor agent) to exchange messages with both broker agent and provider agent.

Provider
• Person or organization that owns a Web service it wants to make available for usage
• Use an agent (i.e provider agent) to exchange messages with broker agent and requestor agent.
• The provider agent is also the software piece which implements the Web service (e.g. mapping towards legacy)

Broker
• Person or organization that puts requestors and providers in contact
  – Providers use brokers to publish Web services
  – Requestors use brokers to discover Web services
• Use an agent (i.e broker agent) to exchange messages with requestor agent and provider agent
Business model
Technologies

Some of the technologies are mandatory for some Web services while optional for other Web services:

HTTP
- Mandatory for RESTful Web services but optional for SOAP Based Web services

XML
- Mandatory for SOAP Based Web Services but optional for RESTful Web services
HTTP

HTTP (HyperText Transfer Protocol)

- Is an application-level protocol for distributed, collaborative, hypermedia information systems
  - HTTP has been in use since 1990
  - HTTP is a request-response protocol
  - HTTP requests relates to resources
    - A resource is any object or service network that can be identified by a URI (Universal Resource Identifier)
HTTP

Client
– A program that establishes connections for the purpose of sending requests

User Agent
– The client which initiates a request (e.g. browser)

• Note
  • A request may pass through several servers
HTTP

Server
- An application program that accepts connections in order to service requests by sending back responses
- A given program may be capable of being both a client and a server
- The role depends on connections
HTTP

- **Origin server**
  - The server on which a given resource resides or is to be created

- **Proxy server**
  - An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients

- **Gateway server**
  - receives requests as if it were the origin server for the requested resource, and forwards the request to another server
  - Is transparent to the client
HTTP

HTTP-message = Request | Response

generic-message = start-line
*(message-header CRLF)
CRLF
[ message-body ]

start-line = Request-Line | Status-Line
**HTTP**

**HEAD**
- retrieve meta-information about a web page, without retrieving the page content (ex: get the date for last modification)

**GET**
- retrieve the page content

**PUT**
- store the enclosed content under the supplied Request-URI

**POST**
- add the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI
  - E.g.
    - Post a message to a mailinglist
    - Extend a database by appending information
    - Transfer a form data
HTTP

DELETE
  ▪ Deletes the page

TRACE
  ▪ Debug

OPTIONS
  ▪ Allows the client to discover the options supported by the server

CONNECT
  ▪ Not used currently
HTTP

The built-in HTTP request methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Request to read a Web page</td>
</tr>
<tr>
<td>HEAD</td>
<td>Request to read a Web page’s header</td>
</tr>
<tr>
<td>PUT</td>
<td>Request to store a Web page</td>
</tr>
<tr>
<td>POST</td>
<td>Append to a named resource (e.g., a Web page)</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove the Web page</td>
</tr>
<tr>
<td>TRACE</td>
<td>Echo the incoming request</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Query certain options</td>
</tr>
</tbody>
</table>
HTTP

The status code response groups.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Information</td>
<td>100 = server agrees to handle client’s request</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>200 = request succeeded; 204 = no content present</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>301 = page moved; 304 = cached page still valid</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>403 = forbidden page; 404 = page not found</td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>500 = internal server error; 503 = try again later</td>
</tr>
</tbody>
</table>
XML

XML is a markup language for documents containing structured information.

XML makes use of tags just like HTML.

- In HTML, both tag semantics (<p> means paragraph) and tag set are fixed.

XML was designed to overcome the limitations of HTML:

- Better support for dynamic content creation and management
- Interactions between programs going further than browser / Web page

W3C recommendation
XML

Main differences between HTTP and XML

• XML was designed to carry data
• XML is not a replacement for HTML
• XML and HTML were designed with different goals
  • XML was designed to describe data and to focus on what data is.
  • HTML was designed to display data and to focus on how data looks.
  • HTML is about displaying information, while XML is about describing information.
• XML is free and extensible (xml tags are not predefined)
XML advantages

- Structure information
- Separate actual data from data representation
- Store data in plain text format
- Share data in software-and-hardware independent way
- Exchange data between incompatible systems
- Create new languages (WAP, WML).
- Platform independent
XML documents
Data objects made of elements
- `<element> content </element>`

Well-formed Documents
- If it obeys to the XML syntax
  - Exp: - All XML elements must have a closing tag
    - The name in an element's end-tag MUST match the element type in the start-tag.
    - All XML elements must be properly nested

Valid document
- A well-formed document is valid if it obeys to the structural rules of the associated DTD or schema document
XML documents

Schema and DTD (Document Type Definition)
- Provide a grammar for a class of documents
- Define the legal elements of an XML document

Namespace
- An XML namespace is a collection of names, identified by a URI reference
- Provides a simple method for qualifying element and attribute names used in XML documents.
XML documents Example

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<book>
    <title>Understanding Web Services</title>
    <author>Eric Newcomer</author>
    <price>39.99</price>
</book>
```
XML

XML processor

- Read XML documents
- Provide access to the content and the structure
- Behaviour described in the XML specifications
- Navigate XML document structure and add, modify, or delete its elements.

- Most popular programming APIs
  - Document Object Model (DOM) from W3C
  - Simple API for XML (SAX) – From XML-DEV mailing list
RESTFul Web Services

1. Introduction
2. Resource Oriented Architecture
3. Resources
4. Properties
5. Tool kits
6. Examples of RESTful Web services
Introduction

- What about using the Web’s basic technologies (e.g. HTTP) as a platform for distributed services?
  
  - This is what is REST about.
Introduction

- REST was first coined by Roy Fielding in his Ph.D. dissertation in 2000.
- It is a network architectural style for distributed hypermedia systems.
- It is not an architecture, but a set of design criteria that can be used to assess an architecture.
Introduction

- REST is a way to reunite the programmable web with the human web.

- It is simple
  - Uses existing web standards
  - The necessary infrastructure has already become pervasive
  - RESTFull web services are lightweight
  - HTTP traverse firewall
Introduction

- RESTful web services are easy for clients to use

- Relies on HTTP and inherits its advantages, mainly
  - Statelessness
  - Addressability
  - Unified interface
The Resource-Oriented Architecture (ROA)

- Is a RESTful architecture
- Provides a commonsense set of rules for designing RESTful web services
Resource-Oriented Architecture

- **Concepts**
  - Resources
    - Resources names (Unified Resource Identifiers-URIs)
    - Resources representations
    - Links between resources

- **Key properties:**
  - Addressability
  - Statelessness
  - Uniform interface
Resources

What’s a Resource?

- A resource is any information that
  - can be named
  - Is important enough to be referenced as a thing in itself
- A resource may be a physical object or an abstract concept
- e.g.
  - a document
  - a row in a database
  - the result of running an algorithm.
Naming:

- Unified Resource Identifier (URI)
  - The URI is the name and address of a resource
  - Each resource should have at least one URI
  - URIs should have a structure and should vary in predictable ways
Resource Representation

- A representation is any useful information about the state of a resource

- Different representation formats can be used (Unlike SOAP based Web services)
  - `plain-text`
  - `JSON`
  - `XML`
  - `XHTML`
  - ....
In most RESTful web services, representations are hypermedia
  i.e. documents that contain data, and links to other resources.
Properties

- **Addressability**
  - An application is addressable if it exposes a URI for every piece of information it serves

- This may be an infinite number of URIs
  - e.g. for search results
    - [http://www.google.com/search?q=jellyfish](http://www.google.com/search?q=jellyfish)
Properties

- **Statelessness**
  - The state should stay on the client side, and be transmitted to the server for every request that needs it.
    - Makes the protocol simpler
    - Ease load balancing
Properties

- **Uniform interface**
  - **HTTP GET:**
    - Retrieve a representation of a resource
  - **HTTP PUT**
    - Create a new resource, where the client is in charge of creating the resource URI: **HTTP PUT** to the new URI
    - Modify an existing resource: **HTTP PUT** to an existing URI
  - **HTTP POST:**
    - Create a new resource, where the server is in charge of creating the resource URI: **HTTP POST** to the URI of the superordinate of the new resource
  - **HTTP DELETE:**
    - Delete an existing resource:
  - **HTTP HEAD:**
    - Fetch metadata about a resource
  - **HTTP OPTIONS:**
    - Lets the client discover what it’s allowed to do with a resource.
Examples of tool kits

- RestLet
- Jersey
Examples of RESTful Web Services

- Examples of existing RESTful web services include:
  - Amazon’s Simple Storage Service (S3) ([http://aws.amazon.com/s3](http://aws.amazon.com/s3))
  - [Twitter](http://twitter.com) is a popular blogging site that uses RESTful Web services extensively.
  - Most other read-only web services that don’t use SOAP
  - Static web sites
  - Many web applications, especially read-only ones like search engines
## Examples of RESTful Web Services

<table>
<thead>
<tr>
<th>Resources</th>
<th>URL</th>
<th>HTTP action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outbound SMS message requests</td>
<td>/outbound/{senderAddress}/requests</td>
<td>GET: read pending outbound message requests</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POST: create new outbound messages request</td>
</tr>
<tr>
<td>Outbound SMS message request and delivery status</td>
<td>/outbound/{senderAddress}/requests/{requestId}</td>
<td>GET: read a given sent message, along with its delivery status</td>
</tr>
<tr>
<td>Inbound SMS message subscriptions</td>
<td>/inbound/subscriptions</td>
<td>GET: read all active subscriptions POST: create new message subscription</td>
</tr>
<tr>
<td>Individual inbound SMS message subscription</td>
<td>/inbound/subscriptions/{subscriptionId}</td>
<td>GET: read individual subscription DELETE: remove subscription and stop corresponding notifications</td>
</tr>
</tbody>
</table>

Table 2. A subset of ParlayREST SMS resources.
Examples of RESTful Web Services

**Figure 4. Sample scenario for SMS handling.**
Illustrative Use Case on conferencing

1. Procedure
2. On conferencing semantics
3. Applying the procedure
The procedure – First Part

- Figure out the data set
- Split the data set into resources
The procedure – Second Part

For each resource:
- Name the resources with URIs
- Identify the subset of the uniform interface that is exposed by the resource
- Design the representation(s) as received (in a request) from and sent (in a reply) to the client
- Consider the typical course of events by exploring and defining how the new service behaves and what happens during a successful execution
On Conferencing semantics

- The conversational exchange of multimedia content between several parties
  - About multimedia
    - Audio, video, data, messaging
  - About participants
    - Any one who wants to participates the conference
On Conferencing semantics

Classification:
- Dial-in / dial-out
- Open/close
- Pre-arranged/ad hoc
- With/without sub-conferencing (i.e. sidebar)
- With/without floor control
On conferencing semantics

- Case considered in the use case
  - Create a service that allows a conference manager to:
    - Create a conference
    - Terminate a conference
    - Get a conference status
    - Add users to a conference
    - Remove users from a conference
    - Change media for a participant
    - Get a participant media
Applying the procedure – First part

1. Data set

- Conferences
- Participants
- Media
Applying the procedure – First part

2. Split the data set into resources
   - Each conference is a resource
   - Each participant is a resource
   - One special resource that lists the participants
   - One special resource that lists the conferences (if we consider simultaneous conferences)
Applying the procedure – Second part

3. Name the resources with URIs

- I’ll root the web service at
  http://www.confexample.com/
- I will put the list of conferences at the root URI
- Each conference is defined by its ID:
  http://www.confexample.com/{confId}/
- A conference participants’ resources are subordinates of the conference resource:
  - The lists of participants:
    http://www.confexample.com/{confId}/participants/
  - Each participant is identified by his/her URI:
    http://www.confexample.com/{confId}/participants/{participantURI}/
### Applying the procedure – Second part

<table>
<thead>
<tr>
<th>Resource</th>
<th>Exposed subset of the uniform interface</th>
<th>Data representation operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Conference</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create: establish a conference</td>
<td>POST: <a href="http://confexample.com/">http://confexample.com/</a></td>
<td>&lt;conference&gt; &lt;description&gt; discuss project &lt;/description&gt; &lt;maxParticipants&gt;10&lt;/maxParticipants&gt; &lt;/conference&gt;</td>
</tr>
<tr>
<td>Read: Get conference status</td>
<td>GET: <a href="http://confexample.com/%7Bconfid%7D">http://confexample.com/{confid}</a></td>
<td>None</td>
</tr>
<tr>
<td>Delete: end a conference</td>
<td>DELETE: <a href="http://confexample.com/%7Bconfid%7D">http://confexample.com/{confid}</a></td>
<td>None</td>
</tr>
<tr>
<td><strong>List of participant(s)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create: Add a participant</td>
<td>POST: <a href="http://confexample.com/%7Bconfid%7D/participants">http://confexample.com/{confid}/participants</a></td>
<td>&lt;participant&gt; <a href="mailto:alice@ericsson.com">alice@ericsson.com</a> &lt;/participant&gt;</td>
</tr>
<tr>
<td>Read: Get a participant status</td>
<td>GET: <a href="http://confexample.com/%7Bconfid%7D/participants/%7BparticipantURI%7D">http://confexample.com/{confid}/participants/{participantURI}</a></td>
<td>None</td>
</tr>
<tr>
<td>Delete: remove a participant</td>
<td>DELETE: <a href="http://confexample.com/%7Bconfid%7D/participants/%7BparticipantURI%7D">http://confexample.com/{confid}/participants/{participantURI}</a></td>
<td>None</td>
</tr>
</tbody>
</table>
Applying the procedure – Second part

1: POST(http://www.confexample.com)
2: 202 accepted(http://www.confexample.com/conf1@confexample.com)

3: 200 OK
4: GET(http://www.confexample.com/conf1@confexample.com)

5: 200 OK

6: POST(http://www.confexample.com/conf1@confexample.com/ participants, bob@ericsson.com)
7: 202 accepted

The server adds the participant(s) to the conference

8: INVITE
9: OK
10: ACK

11: 200 OK
Applying the procedure – Second part

9. What might go wrong?
   • Conference

<table>
<thead>
<tr>
<th>Operation</th>
<th>Server-&gt;Client</th>
<th>Way it may go wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create (POST)</td>
<td>Success: 200 OK</td>
<td>The received request is not correct (e.g. has a wrong body)</td>
</tr>
<tr>
<td></td>
<td>Failure: 400 Bad Request</td>
<td></td>
</tr>
<tr>
<td>Read (GET)</td>
<td>Success: 200 OK</td>
<td>The targeted conference does not exist</td>
</tr>
<tr>
<td></td>
<td>Failure: 404 Not Found</td>
<td></td>
</tr>
<tr>
<td>Delete (DELETE)</td>
<td>Success: 200 OK</td>
<td>The targeted conference does not exist</td>
</tr>
<tr>
<td></td>
<td>Failure: 404 Not Found</td>
<td></td>
</tr>
</tbody>
</table>
Applying the procedure – Second part

9. What might go wrong?
   ▪ Participant(s)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Server-&gt;Client</th>
<th>Way it may go wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create (POST)</td>
<td>Success: 200 OK</td>
<td>• The received request is not correct (e.g. has a wrong body)</td>
</tr>
<tr>
<td></td>
<td>Failure: 400 Bad Request</td>
<td>• The target conference does not exist</td>
</tr>
<tr>
<td></td>
<td>Failure: 404 Not Found</td>
<td></td>
</tr>
<tr>
<td>Read (GET)</td>
<td>Success: 200 OK</td>
<td>• The target conference does not exist</td>
</tr>
<tr>
<td></td>
<td>Failure: 404 Not Found</td>
<td>• The target participant does not exist</td>
</tr>
<tr>
<td>Update (PUT)</td>
<td>Success: 200 OK</td>
<td>• The received request is not correct</td>
</tr>
<tr>
<td></td>
<td>Failure: 400 Bad Request</td>
<td>• The target conference does not exist</td>
</tr>
<tr>
<td></td>
<td>Failure: 404 Not Found</td>
<td>• The target participant does not exist</td>
</tr>
<tr>
<td>Delete (DELETE)</td>
<td>Success: 200 OK</td>
<td>• The target conference does not exist</td>
</tr>
<tr>
<td></td>
<td>Failure: 404 Not Found</td>
<td>• The target participant does not exist</td>
</tr>
</tbody>
</table>
References

• F. Belqasmi, C. Fu, R. Glitho, Services Provisioning in Next Generation Networks: A Survey, IEEE Communications Magazine, December 2011


• Lightweight REST Framework, http://www.restlet.org/


• C. Pautasso and E. Wilde, “Why is the web loosely coupled? A multi-faceted metric for service design”, in Proc. of the 18th World Wide Web Conference, Madrid, Spain (April 2009)