أكاديمية الحكومة الإلكترونية الفلسطينية

The Palestinian eGovernment Academy

www.egovacademy.ps

Tutorial III:

Process Integration and Service Oriented Architectures

Session 5
REST Web Services

Prepared By

Mohammed Melhem

European Commission

TEMPUS

About

This tutorial is part of the PalGov project, funded by the TEMPUS IV program of the Commission of the European Communities, grant agreement 511159-TEMPUS-1-2010-1-PS-TEMPUS-JPHES. The project website: www.egovacademy.ps

Project Consortium:

Birzeit University, Palestine (**Coordinator**)	University of Trento, Italy
Palestine Polytechnic University, Palestine	Vrije Universiteit Brussel, Belgium
Palestine Technical University, Palestine	Université de Savoie, France
Ministry of Telecom and IT, Palestine	University of Namur, Belgium
Ministry of Interior, Palestine	TrueTrust, UK
Ministry of Local Government, Palestine	

Coordinator:
Dr. Mustafa Jarrar
Birzeit University, P.O.Box 14- Birzeit, Palestine
Telfax:+972 2 2982935 mjarrar@birzeit.edu

© Copyright Notes

Everyone is encouraged to use this material, or part of it, but should properly cite the project (logo and website), and the author of that part.

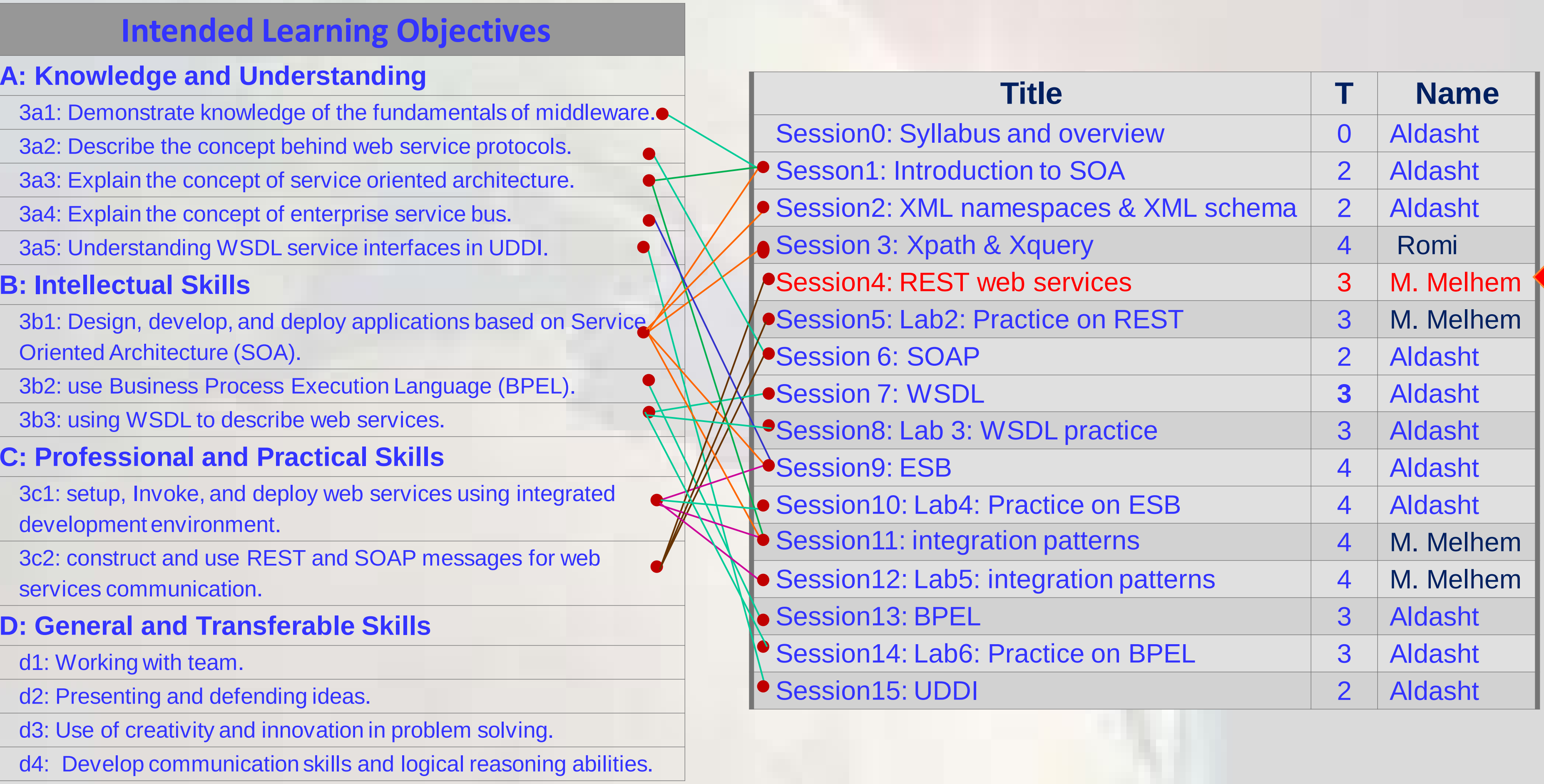
No part of this tutorial may be reproduced or modified in any form or by any means, without prior written permission from the project, who have the full copyrights on the material.

Attribution-NonCommercial-ShareAlike
CC-BY-NC-SA

This license lets others remix, tweak, and build upon your work non-commercially, as long as they credit you and license their new creations under the identical terms.

The Palestinian e-government academy

Tutorial Map



Intended Learning Objectives
A: Knowledge and Understanding
3a1: Demonstrate knowledge of the fundamentals of middleware.
3a2: Describe the concept behind web service protocols.
3a3: Explain the concept of service oriented architecture.
3a4: Explain the concept of enterprise service bus.
3a5: Understanding WSDL service interfaces in UDDI.
B: Intellectual Skills
3b1: Design, develop, and deploy applications based on Service Oriented Architecture (SOA).
3b2: use Business Process Execution Language (BPEL).
3b3: using WSDL to describe web services.
C: Professional and Practical Skills
3c1: setup, Invoke, and deploy web services using integrated development environment.
3c2: construct and use REST and SOAP messages for web services communication.
D: General and Transferable Skills
d1: Working with team.
d2: Presenting and defending ideas.
d3: Use of creativity and innovation in problem solving.
d4: Develop communication skills and logical reasoning abilities.

Title	T	Name
Session0: Syllabus and overview	0	Aldasht
Sesson1: Introduction to SOA	2	Aldasht
Session2: XML namespaces & XML schema	2	Aldasht
Session 3: Xpath & Xquery	4	Romi
Session4: REST web services	3	M. Melhem
Session5: Lab2: Practice on REST	3	M. Melhem
Session 6: SOAP	2	Aldasht
Session 7: WSDL	**3**	Aldasht
Session8: Lab 3: WSDL practice	3	Aldasht
Session9: ESB	4	Aldasht
Session10: Lab4: Practice on ESB	4	Aldasht
Session11: integration patterns	4	M. Melhem
Session12: Lab5: integration patterns	4	M. Melhem
Session13: BPEL	3	Aldasht
Session14: Lab6: Practice on BPEL	3	Aldasht
Session15: UDDI	2	Aldasht

The Palestinian e-government academy

Session Outlines

1. What is REST?
2. Key Principles
3. Fundamental HTTP Concepts
4. Rest examples
5. Design guide lines
6. How to Document REST
7. Case Study
8. Common misuses
9. Code samples (C#, Java)
10. What we have else?
11. Conclusions

What is REST

- **A service designed to embrace the "web" from the ground-up.**
- **Re**presentational **S**tate **T**ransfer
- An architecture style for designing networked application.
  - Introduced by Roy Fielding's PhD Thesis (2000)
- The idea is that, provide a replacement for old and complex integration mechanisms RPC, COBRA, etc. using simple HTTP calls between systems.
  - The web itself, can be viewed as a REST-based architecture.

Key Principles

1. Every element must have it’s ID.
2. Things must be linked.
3. Use standard methods
4. Provide multiple representations
5. Communicate statelessly

Every element must have it’s ID

- Relies on uniform interface URI’s
- Every thing is a resource (Data and Operations)
- Each resource has it a URI
  - Operations, process steps, etc.
  - Individual item
    - http://bzu.edu/course/123
    - http://ritaj.bzu.edu/faculty/1234
  - Collections
    - http://registration.bzu.edu/course/123/students

Things must be linked

- There is **no connection state** interaction is stateless, each request must carry all the information required to complete it, and must not rely on previous operations or requests.
- Two schools
  - The URL King (TUK)[3]
    - Encode all important staff in the URL
  - Hipermedia as the engine of application state (HARTEOAS) [2]

HARTEOAS

- Each URI maps to a single resource
- Each resource may have more than one URI i.e. versions.
- Details:
  - Client takes actions from the representations returned.
  - The media types used for the representations, and the links they may contain.

The Palestinian e-government academy

HATEOAS Example

Example Create record

- Request

```
PUT /Phone HTTP1.1
Host: http://example.com
Content: application/xml
<phone><person>Mohammed</person></phone>
```

- Response

```
201 Created
Location: http//example.com/phones/mohammed
Content: application/xml
….
```

Provide multiple representations

- Rest is not a standard
  - There will never be a W3C recommendation.
  - Often server response is an XML, however other formats can also be used unlike other services i.e SOAP.
  - All data formats are acceptable such as JSON “JavaScript Object Notation”, however using human readable formats is not acceptable.

The Palestinian
e-government
academy

Communicate statelessly

REST services are self contained and each request carries (transfer) all the information (state) that the server needs in order to complete it.

Example: Clients checkout
Client: I'm done, (identification) with a purchase
Server: OK, here it is your total for the items in http://example.com/users/(identification) /basket

Fundamental HTTP Concepts

- Standard communication protocol for interacting with resources and other resources.
- HTTP defines:
  - A standard set of methods
  - Status codes
  - And headers for interaction with resources on the WEB

Standard methods

- Common set of methods to represent CRUD operations, based on HTTP standard verbs and response codes.

Method	Description	CRUD	SAFE	idempotent
Get	Request a specific representation of a resource	Read	YES	YES
PUT	Create/Update a resource with provides representation	UPDATE/CREATE	NO	YES
Delete	Delete the specified resource	DELETE	NO	YES
POST	Submits data to be processed by the identified resource	CREATE/UPDATE	NO	NO
OPTIONS/ HEAD	Returns the methods supported by the identified resource	-	YES	YES

Response Codes

Status Range	Description	Example
100 – 199	Informational	100 continue
200 – 299	Successful	200 OK 201 Created 202 Accepted
300 – 399	Redirection	301 Moved Permanently 304 Not Modified
400 – 499	Client Error	400 Bad Request 401 Client error 404 Not found
500 – 599	Server Error	500 internal server error 501 Not implemented

HTTP Headers

- Used to negotiate behavior between HTTP clients and server.
- Provide built in solutions for important communication concepts like:
  - Redirection
  - Content negotiation
  - Security(Authentication, authorization)
  - Caching
  - Compression

Rest examples

Today most of the online solutions expose their Application programmable interfaces (APIs) as a web services, these services typically exposed as REST, and some of leading companies provide a side WSDL services that easier to parse:

Examples of sites expose their services in REST Format:

- Microsft.com
- Filcker
- Amazon.com
- Facebook.com

Design guide lines

TIPS

- Use logical URL, don’t point to a physical paths.
- Don’t return bulks of data use pagination, and returned results should contain next and previous links.
- Make sure REST services are will documented, and don’t change its return type.
- Always include URI for additional actions “don’t let the client do it by himself/herself”
- GET request should not change state at anytime, use POST/DELETE/PUT for change operations.

How to Document REST

- WSDL and WADL
  - WSDL stands for Web Service Description Language, it is recommended standard by W3C to description SOAP services. WSDL is a flexible language, however it lack for HTTP verbs support, making it a poor choice for REST documentation.
    - WSDP 2.0 support HTTP verbs
  - WADL[4] the Web Application Description Language, an alternative for WSDL to document REST services, it is easy to understand and lighter compared to WSDL, however it lacks WSDL flexibility.

WADL example

```
<application xmlns="http://wadl.dev.java.net/2009/02"
  xmlns:app="http://www.w3.org/2007/app"
  xmlns:atom="http://www.w3.org/2005/Atom">

  <resources base="http://example.org/">
    <resource path="blog/main"
      type="http://www.w3.org/2007/app.wadl#entry_feed">
      <doc title="Main Site"/>
    </resource>
    <resource path="blog/pic"
      type="http://www.w3.org/2007/app.wadl#media_feed">
      <doc title="Pictures"/>
    </resource>
  </resources>
</application>
```

Case STUDY

Lets suppose you are in a university that provide an online services for their students and faculty. One of your services provides supposed to provide following services:

Operation	Description
createStudent	Create a new student
getStudent	Retrieves student information for the authenticated user
deleteStudent	Delete one student
updateStudent	Updates an existing student information
getStudentCourses	Retrieves student registered courses
registerStudentCourse	Register course for an existing student
getCourses	Retrieve all available courses

CASE STUDY

- Identify the resource the service will expose.
  - Student
  - Course
  - registered courses

CASE STUDY

- Analyze the functionality, to address type of resources:
  - An individual student profile
  - A collection of registered courses
  - A collection of courses

CASE STUDY

- Design the URI template
  - start with base url i.e. http://birzeit.edu/
  - May use path component to disambiguate.
  - May use query string for further semantics

    /students/{studentid}

    /students/{studentid}/courses

- Identify status codes to utilized for operations i.e. 201 for new created student record

Design guide lines

- RESTfull interface

Method	URI Template	RPC Equivalent
PUT	students/{studentdI}	createStudent
GET	students/{studentdI}	getStudent
PUT	students/{studentid}	updateStudent
DELETE	students/{studentid}	deleteStudent
GET	students/{studentid}/courses	getStudentCourses
PUT	student/{studentid}/courses/{id}	registerStudentCourse

Case Study [add a phone]

- Request

```
PUT /student/1095080/ HTTP 1.1
Host: birzeit.edu
<Student>
  <name>Karen</name>
  <nationalid>123456789</nationalid>
  <gender>female</gender>
</ Student >
```

- Response

```
201 Created
Location: http://birziet.edu/students/1095080
```

The Palestinian e-government academy

Common misuses

- Exposing implementation details
  - Expose database IDs and database entities
- Ignoring caching
  - It is powerful, increase performance and scalability.
- Ignoring return codes
  - There is more than 200 OK
- Ignoring mime-types
  - Using non-standard mimes
  - Not providing different representations

Code Demo [C#]

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

Code Demo [Java]

```
public static String httpGet(String urlStr) throws IOException {
    URL url = new URL(urlStr);
    HttpURLConnection conn = (HttpURLConnection)
    url.openConnection();
    if (conn.getResponseCode() != 200) {
        throw new IOException(conn.getResponseMessage());
    }
     // Buffer the result into a string
    BufferedReader rd = new BufferedReader( new
    InputStreamReader(conn.getInputStream()));
    StringBuilder sb = new StringBuilder();
    String line;
    while ((line = rd.readLine()) != null) {
        sb.append(line);
    }
    rd.close();
    conn.disconnect();
    return sb.toString();
}
```

What we have else?

- SOAP (Simple Object Access Protocol).
  That been introduced in session1, and will be covered in session 7 in more details.

- OData (Open Data Protocol)
  enables the creation of HTTP-based data services, which allow resources identified using Uniform Resource Identifiers (URIs) and defined in an abstract data model, to be published and edited by Web clients using simple HTTP messages. OData is intended to be used to expose and access information from a variety of sources including, but not limited to, relational databases, file systems, content management systems, and traditional Web sites.

Summary

- REST is an Architecture style
- Platform independent
- Language independent
- Standard based built on top (HTTP).
- Data independent return any type of data, unlike SOAP
- Next session will introduce the SOAP message construction for web services communication.

References

1. Architectural Styles and the Design of Network-based Software Architectures, Roy Thomas Fielding, 2000
2. http://en.wikipedia.org/wiki/HATEOAS
3. Resource Oriented Architecture and REST, Assessment of impact and advantages on INSPIRE, Roberto Lucchi, Michel Millot, European Commission, Joint Research Centre, Institute for Environment and Sustainability, EUR 23397 EN - 2008
4. http://en.wikipedia.org/wiki/Web_Application_Description_Language
5. http://en.wikipedia.org/wiki/SOAP
6. http://www.odata.org/developers/protocols/overview#RelationshipToOtherProtocols

Thanks

Mohammed Melhem