Resource Oriented Architecture

Laboratory of Service Design and Engineering

2011/2012

Outline

- Recap web services
- Designing Read-Only Resource-Oriented
- JAX-RS
- Examples

What’s a Resource?

- A resource is anything that is important enough to be referenced as a thing in itself.
  - It must have at least one URI
- Can two resources be the same?
  - http://www.example.com/software/releases/1.0.3.tar.gz
  - http://www.example.com/software/releases/latest.tar.gz
- Can two URIs designate the same resource?
  - http://www.example.com/sales/2004/Q4
  - http://www.example.com/sales/Q42004
- Can a single URI designate two resources?

HTTP Protocol

- A representation is just some data about the current state of a resource
- A server may provides multiple representations of a resource. How does it figure out which one the client is asking for?
- URI
    - http://www.example.com/stories/104.en
    - http://www.example.com/stories/104.xml
- HTTP header
    - Accept-Language header
    - Accept header

ROA Properties

- Addressability
  - An addressable application exposes a URI for every resource it might serve
- Uniform Interface
  - HTTP methods
- Statelessness
  - Each request is totally disconnected from the others
- Connectness
  - The quality of having links

Designing Read-Only Resource-Oriented Services

How to create a RESTful Web Service?

1. Figure out the data set
2. Split the data set into resources
   - For each kind of resource:
3. Name the resources with URIs
4. Expose a subset of the uniform interface
5. Design the representation(s)
6. Link the resources to each other
7. What's supposed to happen?
8. What might go wrong?

Step 1
Figure out the data set

- What is the data set you'll be exposing?
- Suppose you want to provide a REST interface to http://disi.unitn.it "people" section
- You want to serve data on a particular person
- You want to serve list of faculty members, research members and staff members
- You want to serve the list of papers written by a certain person
- You want to serve the list of papers written by a certain person in year 2008

Step 2
Split the data set into resources

- Predefined one-off resources for especially important aspects of the application
- Top-level directories of other available resources.
- Most services exposes few one-off resources
- Examples
  - list of person working at DISI
  - list of technical members

Step 2
Split the data set into resources

- A resource for every object exposed through the service
- Most services expose a large or infinite number of these resources
- Example
  - Maurizio Marchese
  - Paper -> Science Treks: an autonomous digital library system
  - Technical Report -> OpenKnowledge at work: exploring centralized and decentralized information gathering in emergency contexts

Step 2
Split the data set into resources

- Resources representing the results of algorithms applied to the data set
- This includes collection resources, which are usually the results of queries.
- Most services either expose an infinite number of algorithmic resources, or they don't expose any
- Example
  - List of papers written by Maurizio Marchese in year 2009
  - List of books written by Fabio Casati

Step 3
Name the resources with URIs

- Remember: in a RESTful service the URI contains all the scoping information
- Rules:
  - Use path variables to encode hierarchy
  - http://my.disi.unitn.it/parent/child
  - Put punctuation characters in path variables to avoid implying hierarchy where none exists
  - http://my.disi.unitn.it/parent/child1;child2
  - Use query variables to imply inputs into an algorithm
  - http://my.disi.unitn.it/publications/search?q=2008

Step 3
Name the resources with URIs

- http://my.disi.unitn.it/people
- http://my.disi.unitn.it/people/regular_faculty
- http://my.disi.unitn.it/people/regular_faculty/maurizio_marchese
- http://my.disi.unitn.it/publications/published_papers/year/2008
- http://my.disi.unitn.it/publications/published_papers?year=2008

Step 4
Expose a subset of the uniform interface

- The exposed HTTP methods by a read-only web service are:
  - GET
  - HEAD
    - Retrieve only metadata representation
  - OPTIONS
    - Check which HTTP methods a particular resource supports

Step 5
Design the representation(s)

- What data to send when a client requests a resource
- What data format to use
  - XML, XHTML, JSON, ATOM, and so on
- The representation talks about the state of the resource
- The representation links to other resources (connectedness)

Step 6
Link the resources to each other

- Are our resources designed to be connected?
- How can the client get a list of resources representing the results of algorithms applied to the data set?
- Integrate each new resource into existing resources, using hypermedia links and forms

Step 7
What is supposed to happen?

- Consider the typical course of events
- What GET/HEAD requests does a client send?
- What HTTP request headers should a client send?
- What HTTP response headers should a server send?
- Don't forget the 200 (“OK”) response code!

Step 8
What might go wrong?

- If the server can not fulfill a request, it sends a HTTP error response code
- Some common error response code
  - 400 (“Bad Request”) -> There’s a problem on the client side.
  - 500 (“Internal Server Error”) -> There’s a problem on the server side.
  - 401 (“Unauthorized”) -> The client tried to operate on a protected resource without providing the proper authentication credentials
  - 404 (“Not Found”) -> The client requests a URI that doesn’t map to any resource, the server has no clue what the client is asking for.
  - 409 (“Conflict”) ->The client tries to perform an operation that would leave some resources in an inconsistent state.

Summary

1. Figure out the data set
2. Split the data set into resources
3. For each kind of resource:
4. Name the resources with URIs
5. Expose a subset of the uniform interface
6. Design the representation(s)
7. Link the resources to each other
8. What's supposed to happen?
9. What might go wrong?

Exercise

(Extended version of previous exercise)

- Create RESTFul web services
  - Service accepts integer ID of the student
  - Service should return XML document which represents student
  - Perform CRUD operations
- Use proper HTTP methods for specific task
- Use proper JAX-RS annotation