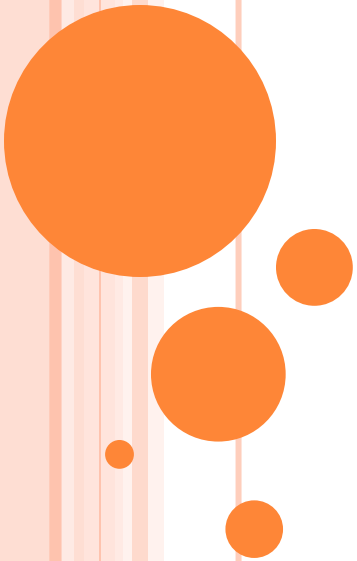

RESTFUL WEB SERVICES

COMP 30220: Distributed Systems

Lecturer: Rem Collier

Email: rem.collier@ucd.ie

RESTful Web Services

- REpresentational State Transfer (REST) is an alternate approach to implementing web services.
  - At its core, REST aims to use existing standards over complex layered architectures (i.e. WSDL/SOAP/HTTP).

- REST is based on the recognition that most interactions with web services are CRUD operations:
  - CRUD = Create, Read, Update, Delete
  - REST combines the use of URIs (Uniform Resource Indicators) with the HTTP API to provide a simple yet effective model of web service interaction:
    - POST = "here's a new resource" (Create)
    - GET = "return the specified resource" (Retrieve)
    - PUT = "here's a revised resource–create if needed" (Update/Create)
    - DELETE = "delete the specified resource" (Delete)

REST Example

- Consider the stock quote service.

- The REST equivalent would look something like this:
  - URL: `http://stock.org/stock/<stock-code>`
  - Use HTTP GET Request because we are querying a value.
  - The response would be data in some format, that can range from plain text, HTML, CSVs, XML, JSON, or even a GIF or JPEG image, …
  - We know the format of the response because of the content-type associated with it.
- Note:
  - There is no WSDL description
  - There is no need to construct a SOAP message
  - All you do is send a GET Request to the specified URL

Passing Data Through REST

- When invoking a RESTful Service, data can be transmitted in one of two ways:
  - **Via the URL**: Key data necessary to identify the resource that is being acted upon is embedded in the URL.
    - Identifying a stock: http://stock.org/stock/IBM
    - Identifying a module: http://www.ucd.ie/modules/COMP30220
  - **Via the HTTP Payload**: for POST and PUT operations, data can also be passed in some agreed format within the payload of the request.
    - Updating the telephone number in a student profile could be modelled as a HTTP PUT to the following URL:

      http://www.ucd.ie/student/13123212

      with a JSON payload:

      ```
      { "telephone" : "012345678" }
      ```

REST and Resources

"The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g. a person), and so on. In other words, any concept that might be the target of an author's hypertext reference must fit within the definition of a resource. A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time."

Roy Fielding

REST and Resources

- The design ethos behind RESTful Services is to mirror the linked resource model of the web.
  - As such, it is expected that the data returned by a GET or POST Request should include URLs to other services that provide additional relevant data.

- Consider a spare parts service offered by Parts Depot Inc.
  - Clients are able to access a RESTful service that returns a list of spare parts that are currently in stock.
    - HTTP GET: http://www.parts-depot.com/parts
  - This service returns an XML document containing a list of spare parts, including URLs that provide details on each part:

```
<?xml version="1.0"?>
<Parts>
      <Part id="00345" href="http://www.parts-depot.com/parts/00345"/>
      <Part id="00346" href="http://www.parts-depot.com/parts/00346"/>
</Parts>
```

REST and Resources

- For example, performing a HTTP GET using the URL for part number 00345 might return:

```
<?xml version="1.0"?>
<Part>
      <Part-ID>00345</Part-ID>
      <Name>Widget-A</Name>
      <Description>This part is used within the frap assembly</Description>
      <Specification href="http://www.parts-depot.com/parts/00345/specification"/>
      <UnitCost currency="USD">0.10</UnitCost>
      <Quantity>10</Quantity>
</Part>
```

- The key idea is that the consumer is able to use the embedded URLs to identify additional web services:
  - This means that the consumer is able to locate and retrieve additional relevant services directly.
  - Obviously, the consumer would need to understand how to interpret the XML data to take advantage of this…

Common Mistakes with REST

- Using the wrong HTTP Operation.
  - There is a tendency to use POST operations over GET and often PUT or DELETE.
- Putting actions into your URL:
  - Example: http://www.ucd.ie/students/create ???
  - This should be POST: http://www.ucd.ie/students!!
- Creating services that are not resources.
  - REST is about CRUD – a GetStockQuote service should be modelled as accessing a stock resource.
- Defining stateful operations.
  - REST is stateless – any operation that requires state to be stored on the server is not REST.
  - E.g. User validation that is managed on the server side.

Typical REST Response Codes

Code	Action	Meaning
200	GET	OK (entity in message body)
	POST	OK (entity in message body)
201	POST	Created (link in Location header)
	PUT	Created (when resource did not exist)
204	DELETE	No Content
401	*	Not Authorised
403	*	Forbidden
404	*	Resource Not Found

http://www.restapitutorial.com/httpstatuscodes.html

Example Interactions

```
POST /parts HTTP/1.1
Host: www.parts-depot.com

<?xml version="1.0"?>
<Part>
    <Part-ID>00345</Part-ID>
    <Name>Widget-A</Name>
    <Description>This part is used within the frap assembly</Description>
    <UnitCost currency="USD">0.10</UnitCost>
    <Quantity>10</Quantity>
</Part>


HTTP/1.1 201 Created
Content-Type: text/xml; charset=utf-8
Location: http://www.parts-depot.com/parts/00345
```

Example Interactions

```
GET /parts HTTP/1.1
Host: www.parts-depot.com
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8

<?xml version="1.0"?>
<Parts>
      <Part id="00345" href="http://www.parts-depot.com/parts/00345"/>
      <Part id="00346" href="http://www.parts-depot.com/parts/00346"/>
</Parts>
```

Example Interactions

```
DELETE /parts/00345 HTTP/1.1
Host: www.parts-depot.com
```

```
HTTP/1.1 204 No Content
```

Implementing REST

- One of the objectives of REST has been to remove the need for complex specifications of web services.
  - Remember REST reduces a web service to CRUD operations that are applied to resources that are uniquely referenced through a URI.
  - The only part of the picture that is not clearly defined is how data is passed to and from these services…
- As we discussed previously, RESTful services can work with any data format, including XML, Json, CSV, …
  - XML has long been acknowledged as the predominant standard for data exchange between software systems.
    - XML schemas can be used to precisely define what is a valid XML document for a given system.
    - XML parsers exist for all main programming languages.
    - But XML is verbose which can make documents more difficult to read (by humans).

JSON

- JSON (JavaScript Object Notation) has emerged as an alternative to XML:
  - Originally developed as a means for representing objects in JavaScript.
  - Rapidly emerged as a data format with the evolution of AJAX (asynchronous JavaScript and XML) due to its more readable format.
  - JavaScript XML parsers transformed XML data into Object structures that were not intuitive to access when compared with JSON.
  - Using JSON as both the object representation and the data format offers a far more intuitive model…
  - This popularity led to JSON being more widely used as a data transfer format…

JSON Syntax

- An **object** is an unordered set of name/value pairs
  - The pairs are enclosed within braces, `{ }`
  - There is a colon between the name and the value
  - Pairs are separated by commas
  - Example: `{ "name": "html", "years": 5 }`

- An **array** is an ordered collection of values
  - The values are enclosed within brackets, `[ ]`
  - Values are separated by commas
  - Example: `["html", "xml", "css"]`

JSON Syntax

- A **value** can be: A string, a number, true, false, null, an object, or an array
  - Values can be nested
- **Strings** are enclosed in double quotes, and can contain the usual assortment of escaped characters
- **Numbers** have the usual C/C++/Java syntax, including exponential (E) notation
  - All numbers are decimal--no octal or hexadecimal
- **Whitespace** can be used between any pair of tokens

Comparison of JSON and XML

- Similarities:
  - Both are human readable
  - Both have very simple syntax
  - Both are hierarchical
  - Both are language independent
  - Both can be used with Ajax

- Differences:
  - Syntax is different
  - JSON is less verbose
  - JSON can be parsed by JavaScript's eval method
  - JSON includes arrays
  - Names in JSON must not be JavaScript reserved words
  - XML can be validated
  - JavaScript is not typically used on the server side

Finally: Hateoas

- Hypermedia As the Engine Of Application State
  - An additional layer that sits on top of REST.
  - Defines a **template structure** for JSON documents that can be "easily" understood by a client.
  - Variants for XML do exist…
  - A **standard** way of expressing resources and the links between them.

```
{
   "_links": {
       "self": { "href": "http://example.org/api/user/rem" },
       "modules": { "href": "http://example.org/modules/rem" }
   },
   "id": "rem",
   "name": "Rem Collier"
}
```

- Resources:
  - HAL Specification: http://stateless.co/hal_specification.html