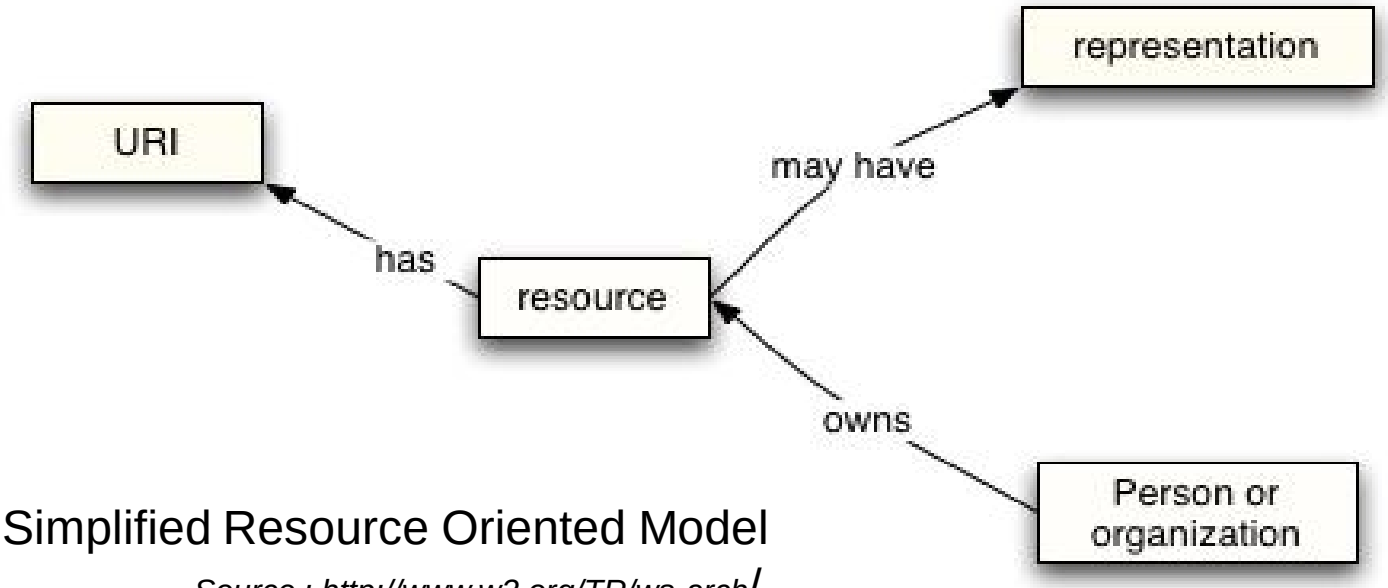
REST

Resource-Oriented Architecture

is a style of software architecture and programming paradigm for designing and developing software in the form of resources with "RESTful" interfaces

- en.wikipedia.org



Simplified Resource Oriented Model

Source : http://www.w3.org/TR/ws-arch/

What is REST?

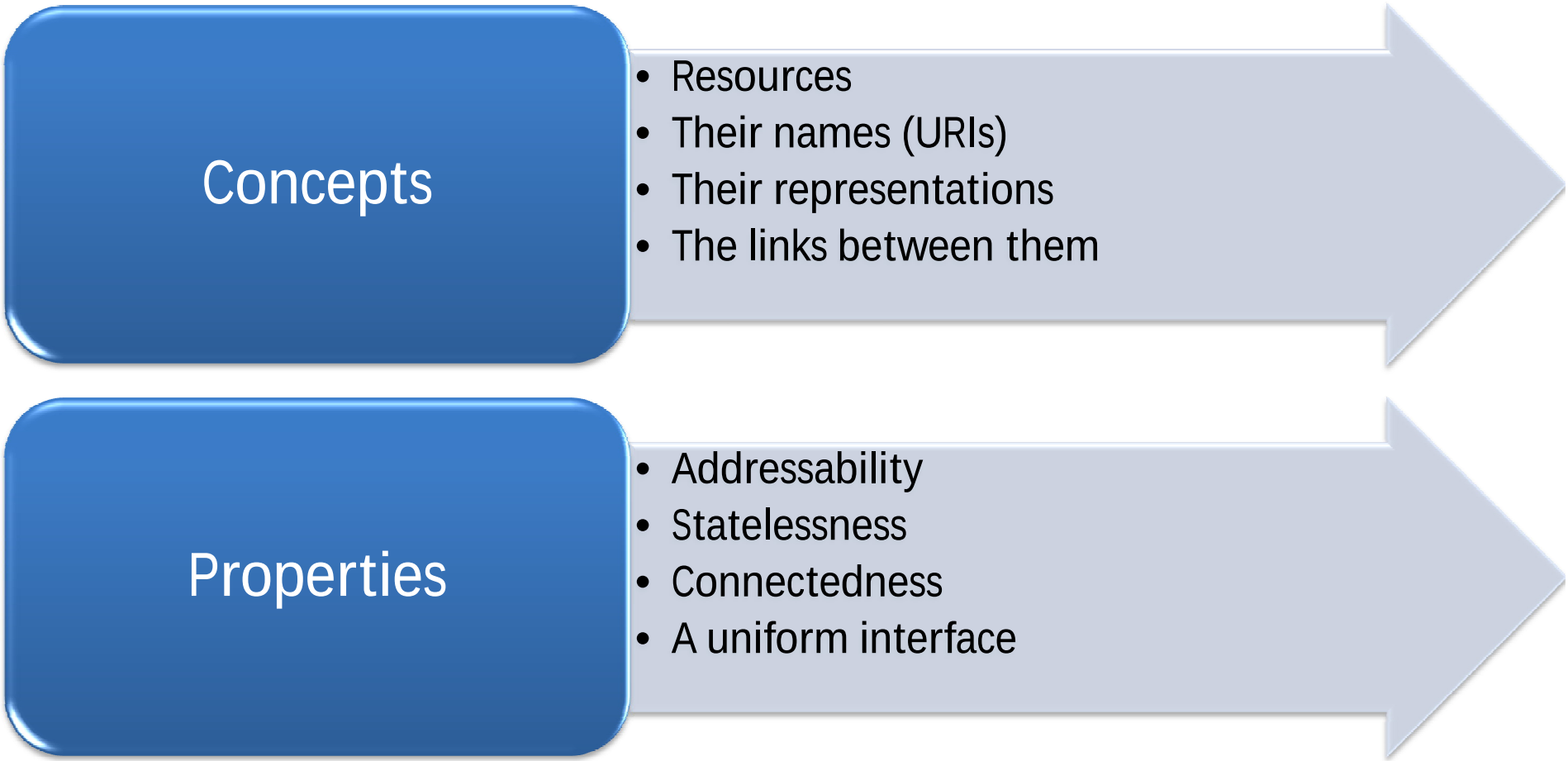
- REpresentational State Transfer
  - PhD by Roy Fielding
  - The Web is the most successful application on the Internet
  - What makes the Web so successful?
- A different way to look at writing Web Services
  - Many say it's the anti-WS-*
  - In my experience, hard for CORBA or WS-* to accept/digest
- REST isn't protocol specific
  - But, usually REST == REST + HTTP

What is REST?

- Addressable Resources
  - Every “thing” should have an ID
  - Every “thing” should have a URI
  - Every “thing” should be referenceable
- Constrained interface
  - Use the standard methods of the protocol
  - HTTP: GET, POST, PUT, DELETE, etc.
- Resources with multiple representations
  - Different applications need different formats
  - Different platforms need different representations (AJAX + JSON)
- Communicate statelessly
  - Stateless application scale

The Resource Oriented Architecture

ROA – Concepts & Properties

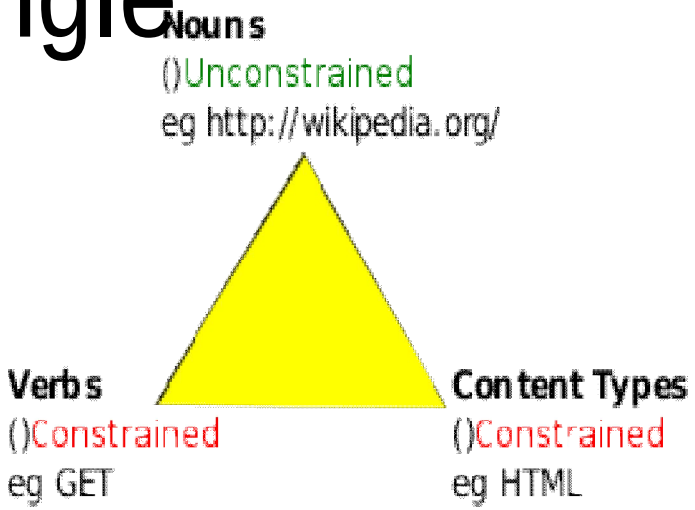


REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation data	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Source : http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#sec_5_2

REST Triangle



Nouns: Know Your URIs

- is an identifier for a resource
- Generally a URL, URN, URI
- Eg http://example.com/=/model/person/id/157

Verbs: Know Your CRUD

- common changes made to data: Create, Read, Update, and Delete
  - GET performs a read operation
  - POST performs a create operation
  - PUT performs an update operation
  - DELETE performs a delete operation
- Not mandatory to have all the operations in a RESTful web service

Content Types: Know Your MIME

- Format of data used in the RESTful application
- "Content-Type" header

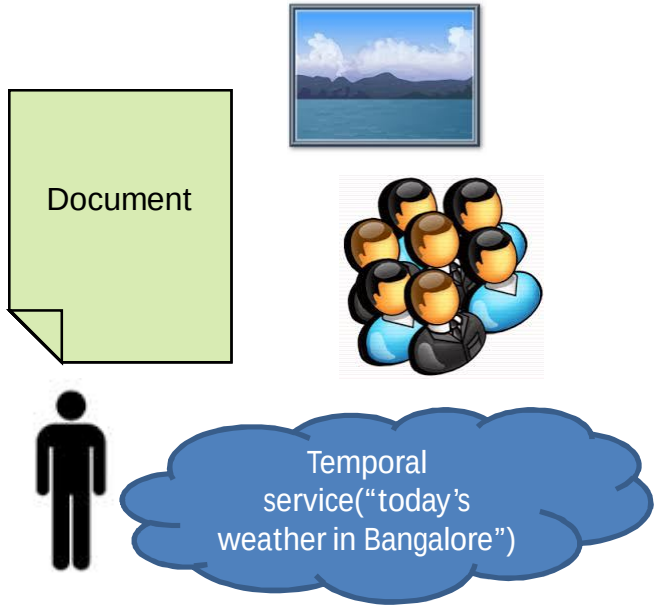
What's a Resource ?

- key abstraction of information in REST

  - any item of interest :
    document/image, non virtual object
    etc.

- conceptual mapping to a set of entities

- *A resource R is a temporally varying membership function MR(t), which for time t maps to a set of entities, or values, which are equivalent.*
- *The values in the set may be resource representations and/or resource identifiers.*



Source : http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#sec_5_2

URIs

Each resource is identified by one or more Uniform Resource Identifiers (URIs)

Twitter: https://dev.twitter.com/docs/api

Facebook: http://developers.facebook.com/docs/reference/api/

LinkedIn: https://developer.linkedin.com/apis

URIs – Criteria for good URI

- Nouns, not verbs
- Short (as possible)
- Hackable ‘up the tree’.
  - remove the leaf path and get an expected page back
- Meaningful
- Predictable. Human-guessable. Readable
- Consistent
- Stateless
- Return a representation (e.g. XML or json) based on the request headers
- Tied to a resource. Permanent
- Report canonical URIs.
- Uses name1=value1;name2=value2 (aka matrix parameters) when filtering collections of resources.

Addressability

- Every “thing” has a URI
  `http://sales.com/customers/323421`
  `http://sales.com/customers/32341/address`

- From a URI we know
  - The protocol (How do we communicate)
  - The host/port (Where it is on network)
  - The resource path(What resource are we communicating with

Addressability

- Its standardized and well-known
  - Anybody that has used a browser understands URIs
  - Java EE has no standard addressability for components. Isn't that a portability headache?
- Linkability
  - Support finds a problem? Have them email you a URI that reproduces the problem
  - Resource representations have a standardized way of referencing other resource representations
  - Representations have a standardized way to compose themselves:

```
<order id="111">
  <customer>http://sales.com/customers/32133</customer>
  <order-entries>
     <order-entry>
        <quantity>5</quantity>
        <product>http://sales.com/products/111</product>
…
```

Describing a URI

- Human readable URIs: Desired but not required
- URI Parameters

http://sales.com/customers/323421

/customers/{customer-id}

- Query parameters to find other resources

http://sales.com/customers?zip=02115

- Matrix parameters to define resource attributes

http://sales.com/customers;zip=02115;state=MA

Constrained, Uniform Interface

- Hardest thing for those with CORBA and/or WS-* baggage to digest
- The idea is to have a well-defined, fixed, finite set of operations
  - Resources can only use these operations
  - Each operation has well-defined, explicit behavior
  - In HTTP land, these methods are GET, POST, PUT, DELETE
- How can we build applications with only 4+ methods?
  - SQL only has 4 operations: INSERT, UPDATE, SELECT, DELETE
  - JMS has a well-defined, fixed set of operations
  - Both are pretty powerful and useful APIs with constrained interfaces

Implications of a Uniform Interface

- Intuitive
  - You know what operations the resource will support
- Predictable behavior
  - GET - readonly and idempotent. Never changes the state of the resource
  - PUT - an idempotent insert or update of a resource. Idempotent because it is repeatable without side effects.
  - DELETE - resource removal and idempotent.
  - POST - non-idempotent, “anything goes” operation
- Clients, developers, admins, operations know what to expect
  - Much easier for admins to assign security roles
  - For idempotent messages, clients don’t have to worry about duplicate messages.

Implications of a Uniform Interface

- Simplified
  - Nothing to install, maintain, upgrade
  - No stubs you have to generate distribute
  - No vendor you have to pay big bucks to
- Platform portability
  - HTTP is ubiquitous. Most (all?) popular languages have an HTTP client library
  - CORBA, WS-*, not as ubiquitous
  - (We'll talk later about multiple representations and HTTP content negotiation which also really helps with portability)
- Interoperability
  - HTTP a stable protocol
  - WS-*, again, is a moving target
  - Ask Xfire, Axis, and Metro how difficult Microsoft interoperability has been
  - Focus on interoperability between applications rather focusing on the interoperability between vendors.

Implications of Uniform Interface

- Familiarity
  - Operations and admins know how to secure, partition, route, and cache HTTP traffic
  - Leverage existing tools and infrastructure instead of creating new ones
- Easily debugged
  - How cool is it to be able to use your browser as a debugging tool!

Designing with a Uniform Interface

```
public interface OrderEntryService {

   void submitOrder(Order order);
   Order[] getOrders();
   void updateOrder(Order order);
   void cancelOrder(int orderId);
   Order[] getCustomerOrders(Customer customer);
   double calculateAverageSale();
}

public interface CustomerService {

   void createCustomer(Customer cust);
   void deleteCustomer(int custId);
   Customer[] getCustomers();
   Customer findCustomer(String first, String last);
}
```

Designing services with a Uniform Interface

- When in doubt, define a new resource
- /orders
  - GET - list all orders
  - POST - submit a new order
- /orders/{order-id}
  - GET - get an order representation
  - PUT - update an order
  - DELETE - cancel an order
- /orders/average-sale
  - GET - calculate average sale
- /customers
  - GET - list all customers
  - POST - create a new customer
- /customers/{cust-id}
  - GET - get a customer representation
  - DELETE- remove a customer
- /customers/{cust-id}/orders
  - GET - get the orders of a customer

Resources with Multiple Representations

- Through URIs and the uniform interface we exchange data
- HTTP allows the client to specify the type of data it is sending and the type of data it would like to receive
- Depending on the environment, the client negotiates on the data exchange
  - An AJAX application may want JSON
  - A Ruby application my want the XML representation of a resource
  - A server may want to serve up a CSV, MS Excel, or PDF representation of a resource

Resources with Multiple Representations

- HTTP Headers manage this negotiation
  - CONTENT-TYPE: specifies MIME type of message body
  - ACCEPT: comma delimited list of one or more MIME types the client would like to receive as a response
  - In the following example, the client is requesting a customer representation in either xml or json format

```
GET /customers/33323
ACCEPT: application/xml,application/json
```

- Preferences are supported and defined by HTTP specification

```
GET /customers/33323
ACCEPT: text/html;q=1.0,
        application/json;q=0.5;application/xml;q=0.7
```

Resources with Multiple Representations

- Internationalization can be negotiated to
  - CONTENT-LANGUAGE: what language is the request body
  - ACCEPT-LANGUAGE: what language is desired by client

```
GET /customers/33323
ACCEPT: application/xml
ACCEPT-LANGUAGE: en_US
```

Resources with Multiple Representations

- This data exchange pattern already exists in many applications
  - Swing->RMI->Hibernate
  - Hibernate objects exchanged to and from client and server
  - Client modifies state, uses entities as DTOs, server merges changes
    - No different than how REST operates
  - No reason a RESTFul webservice and client can't exchange Java objects!

Implications of Multiple Representations

- Evolvable integration-friendly services
  - Common consistent location (URI)
  - Common consistent set of operations (uniform interface)
  - Slap on an exchange formats as needed
- Built-in service versioning
  - Add newer exchange format as an additional MIME type supported
  - application/xml;version=1.0
  - Application/xml;version=1.1
- Internationalization becomes easy for clients
  - Most browsers can configure default ACCEPT-LANGUAGE

Statelessness

- A RESTFul application does not maintain sessions/conversations on the server
- Doesn't mean an application can't have state
- REST mandates
  - That state be converted to resource state
  - Conversational state be held on client and transferred with each request
- Sessions are not linkable
  - You can't link a reference to a service that requires a session
- A stateless application scales
  - Sessions require replication
  - A simplified architecture is easier to debug
- Isolates client from changes on the server
  - Server topology could change during client interaction
  - DNS tables could be updated
  - Request could be rerouted to different machines

REST in Conclusion

- REST answers questions of
  - Why does the Web scale?
  - Why is the Web so ubiquitous?
  - How can I apply the architecture of the Web to my applications?
- REST is tough to swallow
  - Make you rethink how you do things
  - Those with CORBA/WS-* baggage will resist (sometimes violently)
- Promises
  - Simplicity
  - Interoperability
  - Platform independence
  - Change resistance

Spring MVC + REST

RESTFul Web Services using Spring MVC

Content Negotiation

- A URL suffix (path extension)
  - http://...accounts.json to indicate JSON format.
- Or a URL parameter can be used. By default it is named `format`
  - http://...accounts?format=json.
- Or the HTTP Accept header property will be used (which is actually how HTTP is defined to work
  - Not always convenient to use - especially when the client is a browser)

ViewResolver Vs HttpMessageConverter

Handler	When to use
ViewResolvers	Resolve a view or the content you are trying to serve is in elsewhere (e.g. a controller method that returns the string "index" that will be mapped to the index.xhtml page)
HttpMessageConverters	when you use the @ResponseBody annotations for returning the response directly from the method without mapping to an external file

References

- http://spring.io/guides/tutorials/rest/
  - Step by step procedure to build a full-fledged Spring RESTful web application