RESTFUL Web services and their use in telecommunications (Chapter 11)

Dr. Fatna Belqasmi, PhD. Ericsson Canada

Outline

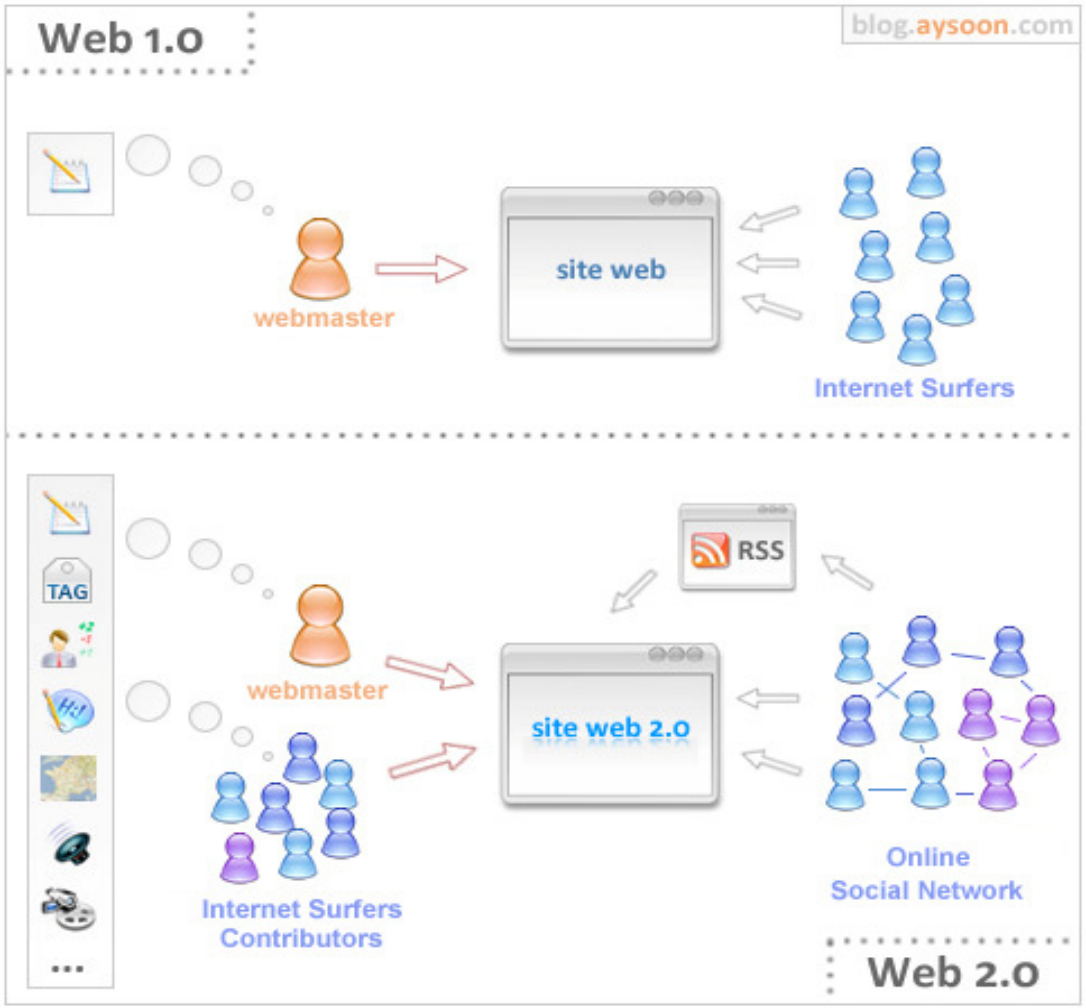
- Why RESTFUL Web services?
- Essentials of RESTFUL Web services
- Using RESTFUL Web services for telecommunications

Why RESTFUL Web services?

- What is Web 2.0?
- How the web works?
- How Big Web Services Works?
- What is REST and why we need it?

What is Web 2.0?

- 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.



What is Web 2.0?

Web 1.0	Web 2.0
– Human web	– Programmable web
– Is about HTML	– Is about XML
– Is about client-server	– Is about peer-to-peer
– Is about reading	– about writing
– Is about companies	– Is about communities
–Is about home pages	– Is about blogs
– Is about owning	– Is about sharing
– Is about services sold over the web	– Is about web services
……	…..

How the web works?

Example web server

- The HTTP client:
  - Connects to the server.
  - Sends the server a method (“GET”) and a path to the resource (“/hello.txt”).
- The server sent back the contents of the requested document.

Client request	Server response
GET /hello.txt HTTP/1.1	200 OK
Host: www.example.com	Content-Type: text/plain
	Hello, world!

How the web works?

- HTTP characteristics
  - a request-response protocol
  - Statelessness
  - Scalability
  - Addressability
  - Cachability
  - Unified interface

How the web works?

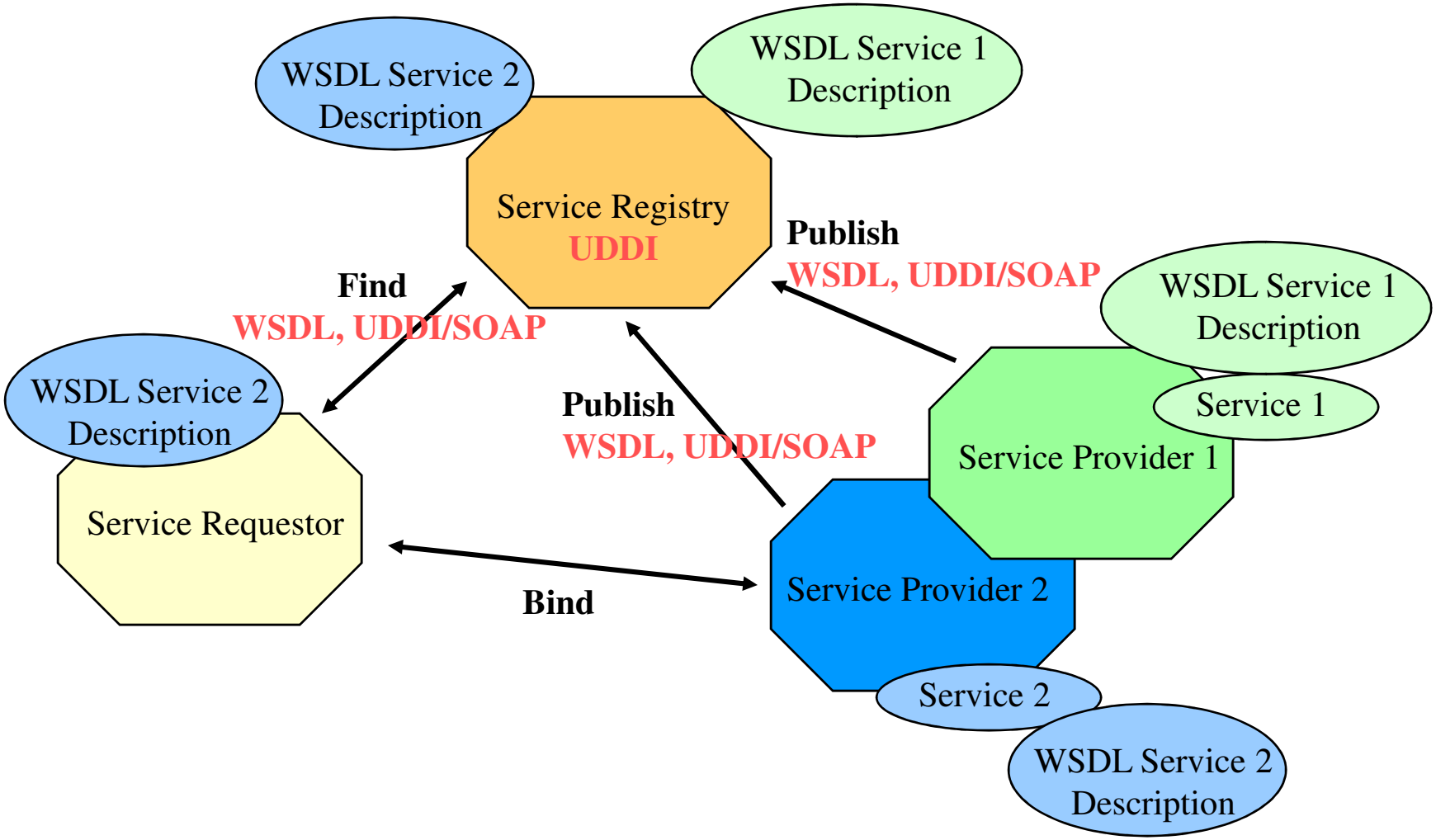
- HTTP methods (RFC 2616)

Safe Methods	Retrieve information
GET	retrieve information identified by the Request-URI
HEAD	retrieve meta-information information identified by the Request-URI
Idempotent Methods	The result is the same if execute once or many times
GET, HEAD	
PUT	store the enclosed entity under the supplied Request-URI
DELETE	delete the resource identified by the 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

How Big Web Services Works?

- ‘Big’ web services are modular programs that can be discovered and invoked over a network.
- They rely on a stack of technologies including XML, SOAP and WSDL.
- The SOAP messages are usually sent across the network using HTTP, although other bindings are possible.

How Big Web Services Works?



How Big Web Services Works?

- Complex
  - Every new layer creates failure points, interoperability, and scalability problems.
  - Many SOAP extensions
  - Clients need to support SOAP
- No unified interface
  - Use Remote Procedure Call (RPC)
  - The method is sent in the SOAP message body
  - SOAP messages are sent using HTTP POST
- All the requests to a given WS are sent to the same URI

```
<env:Envelope
xmlns:env="http://schemas.xmlsoap.org/soap
/envelope/">
  <env:Header />
    <env:Body>
    <startConference xmlns="http://com/conf">
      <str1>alice@ericsson.com</str1>
      <str2>bob@ericsson.com</str2>
      <str3>charles@ericsson.com</str3>
    </startConference>
  </env:Body>
  </env:Envelope>
```

What is REST and why we need it?

- What about using the Web's basic technologies as a platform for distributed services?
  - This is what is REST about.

What is REST and why we need it?

- 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 asses an architecture
- It is not a standard, but uses standards
  - e.g. HTTP, XML, HTML

What is REST and why we need it?

- 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
- RESTFul web services are easy for clients to use
- Relies on HTTP and inherits its advantages, mainly
  - Statelessness
  - Addressability
  - Unified interface

What is REST and why we need it?

RESTFul web services	Big Web Services
• Simple and lightweight • Easy to develop • The method information is given in the URI (i.e. is the HTTP method) • Scoping information is given in the URI • Use HTTP – No extra envelope (except for HTTP) – Can be seen as a 'postcard' • Closer in design and philosophy to the web	• Complex • Harder to develop (requires tools) • The method is given in the request body • Scoping information is given in the request body • Use SOAP/HTTP – +SOAP envelope – Can be seen as a 'letter' inside an envelope

Essentials of RESTFUL Web services

- Resource Oriented Architecture (ROA)
- Tools
- Examples of existing RESTFul web services

Resource-Oriented Architecture

- The Resource-Oriented Architecture (ROA)
  - Is a RESTful architecture
  - Provides a commonsense set of rules for designing RESTful web services
- ROA concepts
  - Resources
  - Resources names (Unified Resource Identifiers-URIs)
  - Resources representations
  - Links between resources
- ROA Properties:
  - Addressability
  - Statelessness
  - Connectedness
  - 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.

- 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
  - *plain-text*
  - *JSON*
  - XML
  - XHTML
  - …

- In most RESTful web services, representations are hypermedia
  - i.e. documents that contain data, and links to other resources.

ROA properties

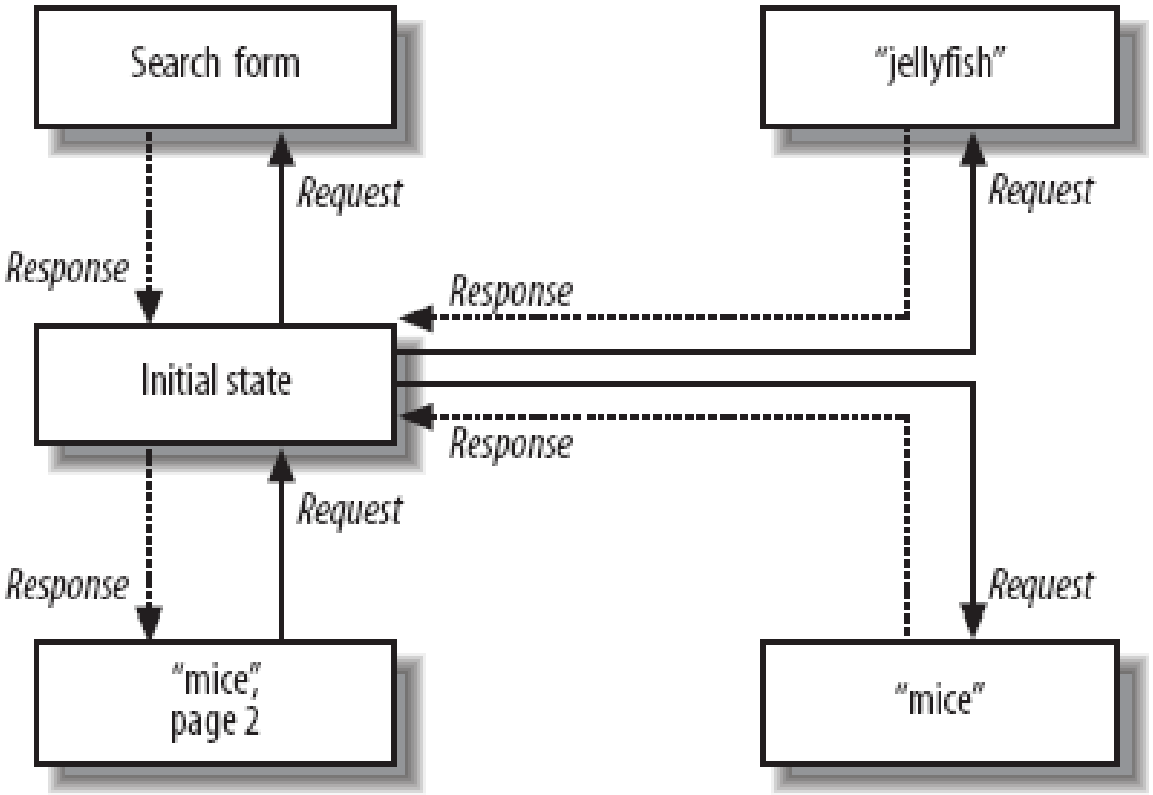
- Addressability
  - An application is addressable if it exposes a URI for every piece of information it serves
  - This may be an infinite number of URIs
    - e.g. for search results
      - *http://www.google.com/search?q=jellyfish*

ROA properties

- Statelessness
  - The state should stay on the client side, and be transmitted to the server for every request that needs it.
- Statelessness
  - Makes the protocol simpler
  - Ease load balancing
  - Ease access to any resource (for client)
- The most common way to break the HTTP intrinsic statelessness is to use HTTP sessions.

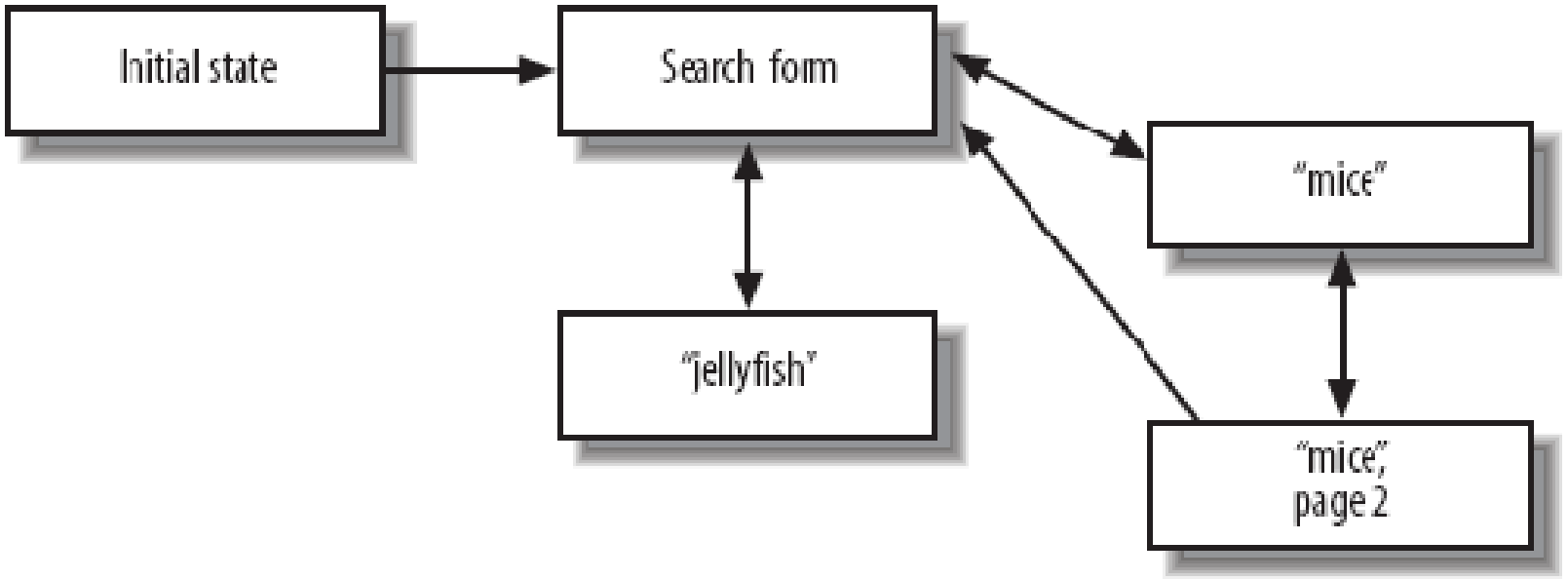
ROA properties

- A stateless search engine



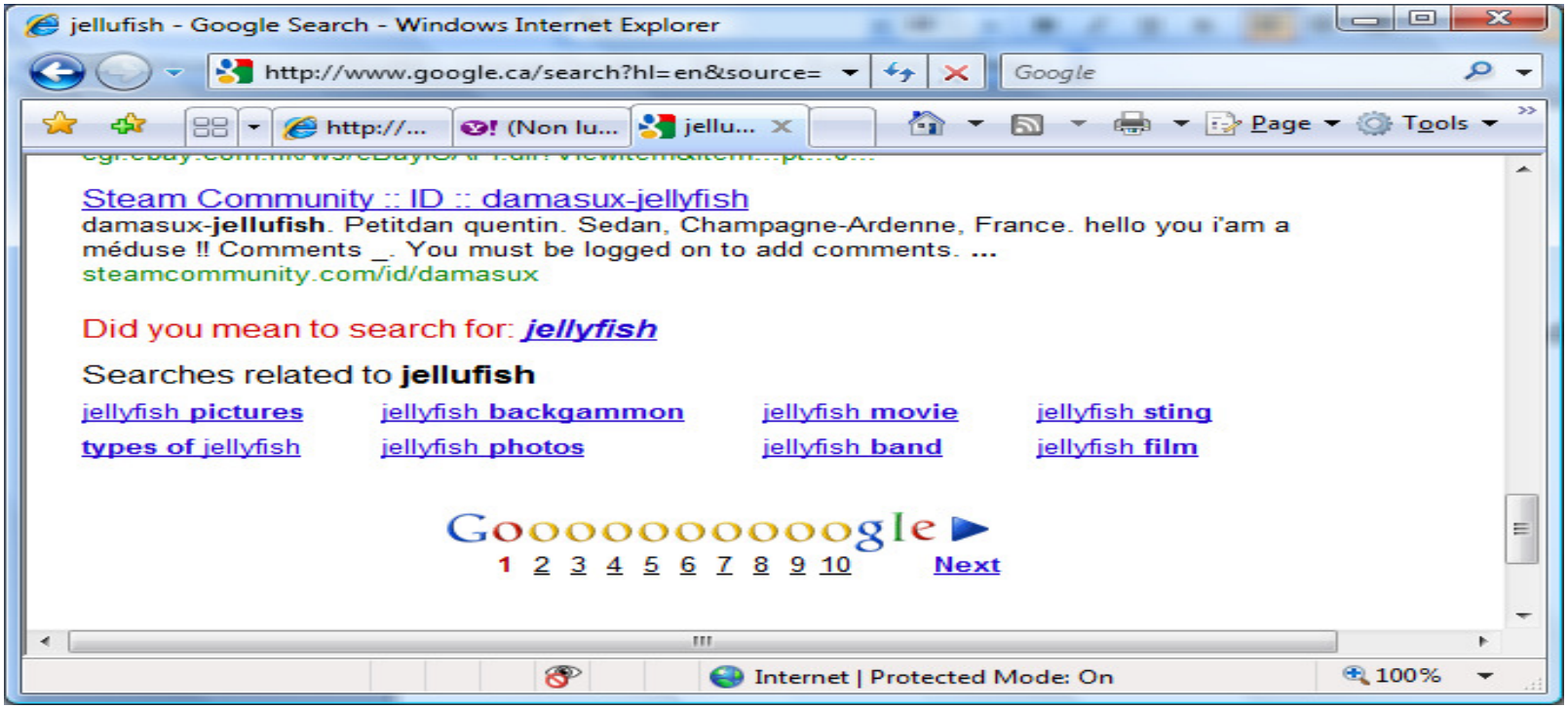
ROA properties

- A stateful search engine



ROA properties

- Connectedness
  - e.g. when searching google, you get
    - Some search results, and a
    - A set of internal links to other pages



ROA 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.

ROA properties

- *PUT and POST actions*

	PUT to a new resource	PUT to an existing resource	POST
/weblogs	N/A (resource already exists)	No effect	Create a new weblog
/weblogs/myweblog	Create this weblog	Modify this weblog's settings	Create a new weblog entry
/weblogs/myweblog/entries/1	N/A (how would you get this URI?)	Edit this weblog entry	Post a comment to this weblog entry

ROA properties

- Safety and Idempotence
  - GET and HEAD requests are *safe*.
  - GET, HEAD, PUT and DELETE requests are *idempotent*.
  - POST is neither safe nor idempotent.
- Why safety and idempotence matter
  - They let a client make reliable HTTP requests over an unreliable network.
- Why the Uniform Interface Matters
  - Any RESTFul service is as similar as any web site
  - No need to learn how each service expected to receive and send information.

Tools

- Techniques
  - HTTP Servlet
  - Ajax
- APIs
  - HTTP Servlet API
  - RestLet
  - JSR 311 API for RESTful web service (JAX-RS or Jersey)
  - *XMLHTTPRequest* API

Existing services

- Examples of existing RESTful web services include:
  - Amazon's Simple Storage Service (S3) (*http://aws.amazon.com/s3*)
  - Services that expose the Atom Publishing Protocol (*http://www.ietf.org/html.charters/atompub-charter.html*) and its variants such as GData (*http://code.google.com/apis/gdata/*)
  - Most of Yahoo!'s web services (*http://developer.yahoo.com/*)
  - Twitter 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

Using RESTFUL Web services for telecommunications

- The procedure to create a RESTFul web service
- Illustrative use case

The procedure 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) accepted from the client
6. Design the representation(s) served to the client
7. Integrate this resource into existing resources, using hypermedia links and forms
8. Consider the typical course of events: what's supposed to happen?
9. Consider error conditions: what might go wrong?

Illustrative use case

- Use case
  - Create a service that allows users to
    - Create a conference
    - Stop a conference
    - Change media for 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

Illustrative use case

1. Figure out the data set
   - Conferences, along with related media and participants

2. Split the data set into resources
   - One special resource that lists the conferences
   - One special resource that lists the participants
   - Each conference is a resource
   - Each participant is a resource

   - In this example, I will not consider media as a resource, but as a conference/participant property

Illustrative use case

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}/

Illustrative use case

4. Expose a subset of the uniform interface

Resource	**Operation CRUD**	**HTTP action**	**Req Body**	**Resp Body**
Conference	Create: establish a conference	POST: http://confexample.com/	YES	YES
	Read: Get conference status	GET: http://confexample.com/{confId}	NO	YES
	Update: Change media for conference	PUT: http://confexample.com/{confId}	YES	NO
	Delete: terminate a conference	DELETE: http://confexample.com/{confId}	NO	NO

Why not to simply use HTML forms to manage a conference?

Illustrative use case

4. Expose a subset of the uniform interface

Resource	**Operation CRUD**	**HTTP action**	**Req Body**	**Resp Body**
Participant(s)	Create: Add participant(s)	POST: http://confexample.com/{confId}/participants	YES	YES
	Read: Get information about a participant	GET: http://confexample.com/{confId}/participants/{participantId}	NO	YES
	Update: Change media for a participant	PUT: http://confexample.com/{confId}/participants/{participantId}	YES	NO
	Delete: delete a participant	DELETE: http://confexample.com/{confId}/participants/{participantId}	NO	NO

Illustrative use case

5-6-7. Design the representation(s) accepted from/served to the client

- Create conference request body:

```
<Participants>
   <Participant>alice@ericsson.com</Participant>
   <Participant>bob@ericsson.com</Participant>
   <Participant>charles@concordia.ca<Participant>
</Participants>
<Media>audio</Media>
```

- Create conference Accept response body:

```
http://www.confexample/{confId}
```

Illustrative use case

5-6-7. Design the representation(s) accepted from/served to the client

- Get conference status response body:

```
<Participants>
   <Participant media="video">alice@ericsson.com</Participant>
   <Participant>bob@ericsson.com</Participant>
   <Participant>charles@concordia.ca<Participant>
</Participants>
<Media>audio</Media>
```

- PUT: change media for a conference request body:

```
<Media>video</Media>
```

Illustrative use case (steps 5-6-7)

- Add participant(s) request body:

```
<Participants>
   <Participant media="audio">alice@ericsson.com</Participant>
   <Participant media="video">bob@ericsson.com</Participant>
</Participants>
```

- Add participant OK response body:

```
<Participants>
   <Participant>
      <uri>alice@ericsson.com</uri>
      <link>http://confexample.com/{confId}/participants/alice@ericsson.com</link>
  </Participant>
   <Participant>
      <uri>bob@ericsson.com</uri>
      <link>http://confexample.com/{confId}/participants/bob@ericsson.com</link>
  </Participant>
</Participants>
```

Illustrative use case

5-6-7. Design the representation(s) accepted from/served to the client

- Get participant status response body:

```
<Participant media="audio">alice@ericsson.com</Participant>
```

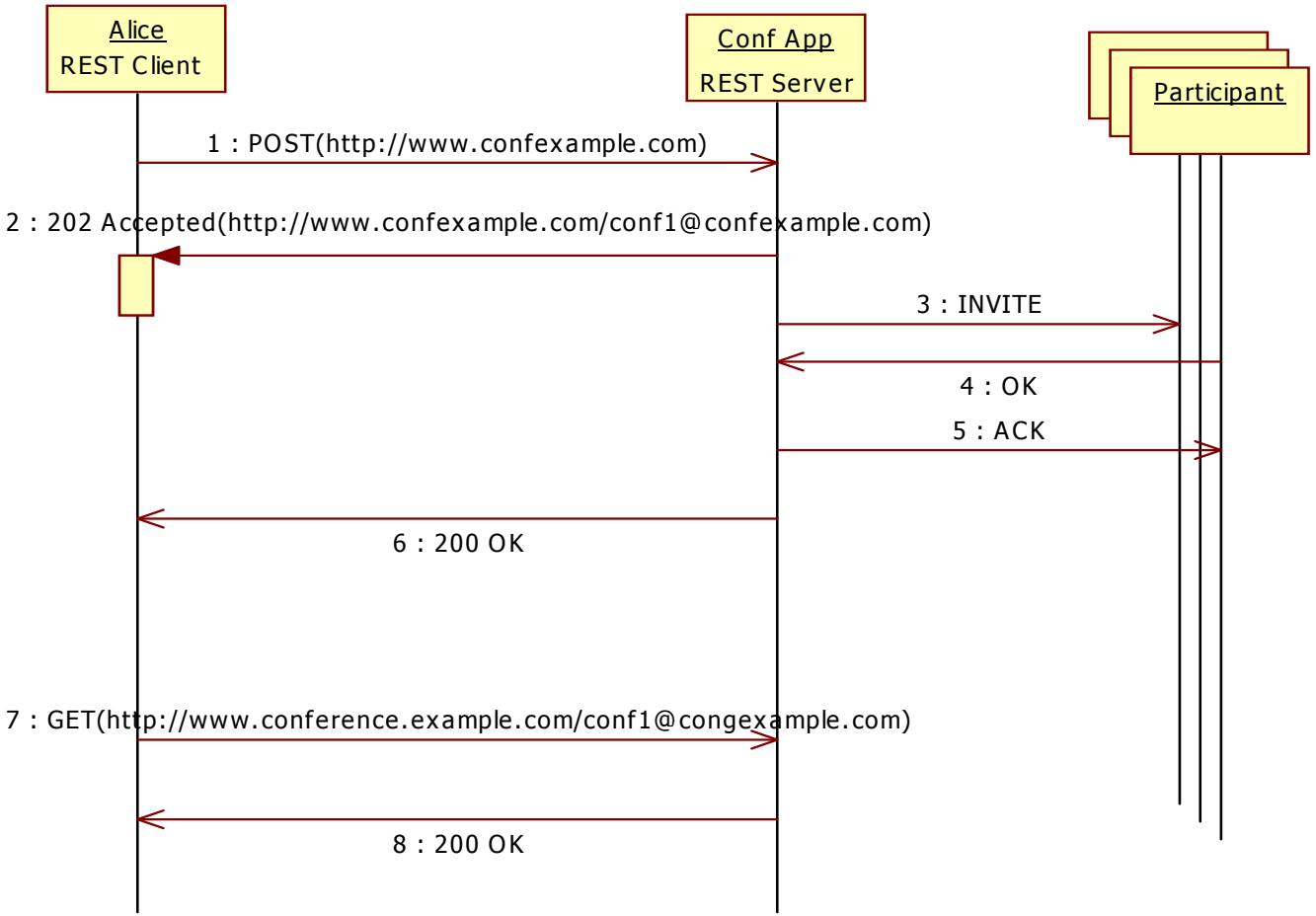
- PUT: change media for a participant request body:

```
<Media>video</Media>
```

Illustrative use case

8. What is supposed to happen?

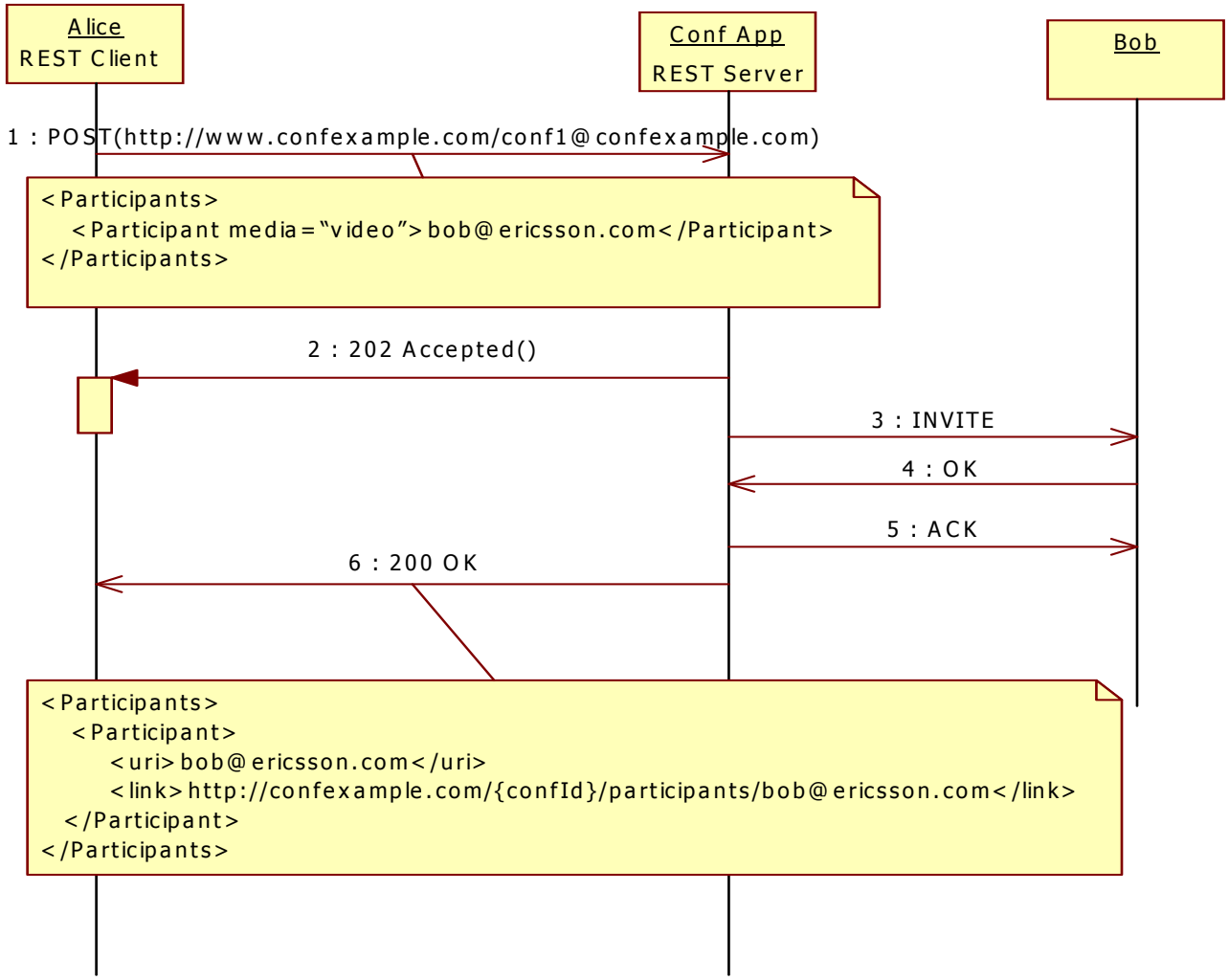
Create conference



Illustrative use case

8. What is supposed to happen?

Add participant



The procedure to create a RESTFul web service

9. What might go wrong?
   - Conference

Operation	**Server->Client**	**Way it may go wrong**
Create (POST)	Success: 200 OK Failure: 400 Bad Request	The received request is not correct (e.g. has a wrong body)
Read (GET)	Success: 200 OK Failure: 404 Not Found	The targeted conference does not exist
Update (PUT)	Success: 200 OK Failure: 400 Bad Request Failure: 404 Not Found	• The received request is not correct (e.g. has a wrong body) • The target conference does not exist
Delete (DELETE)	Success: 200 OK Failure: 404 Not Found	The targeted conference does not exist

Illustrative use case

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

Operation	**Server->Client**	**Way it may go wrong**
Create (POST)	Success: 200 OK Failure: 400 Bad Request Failure: 404 Not Found	• The received request is not correct (e.g. has a wrong body) • The target conference does not exist
Read (GET)	Success: 200 OK Failure: 404 Not Found	• The target conference does not exist • The target participant does not exist
Update (PUT)	Success: 200 OK Failure: 400 Bad Request Failure: 404 Not Found	• The received request is not correct • The target conference does not exist • The target participant does not exist
Delete (DELETE)	Success: 200 OK Failure: 404 Not Found	• The target conference does not exist • The target participant does not exist

References

- L. Richardson and S. Ruby, “RESTful Web Services”, O’ Reilly & Associates, ISBN 10: 0-596-52926-0, May 2007
- Lightweight REST Framework, http://www.restlet.org/
- JSR 311: JAX-RS: The JavaTM API for RESTful Web Services, online at: http://jcp.org/en/jsr/detail?id=311
- C. Pautasso, O. Zimmermann, and F. Leymann, “RESTful Web Services vs. “Big”Web Services: Making the Right Architectural Decision”, In Proceedings of the 17th International World Wide Web Conference, pages 805–814, Beijing, China, April 2008, ACM Press.
- 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)
- C. Pautasso, “Composing restful services with jopera”, in A. Bergel and J. Fabry, editors, Software Composition, volume 5634, Lecture Notes in Computer Science, pages 142–159. Springer, 2009.
- Andreas Kamilaris, “A Lightweight Resource-Oriented Application Framework for Wireless Sensor Networks”, Master Thesis, April 2009