Lesson 15 – SOA with REST (Part II)

Service Oriented Architectures Security

Module 3 - Resource-oriented services

Unit 1 – REST

Ernesto Damiani

Università di Milano

REST Design Tips

1. Understanding GET vs. POST vs. PUT
2. Multiple Representations
   - Content-Type Negotiation
3. Exception Handling
   - Idempotent vs. Unsafe

POST vs. GET

- GET is a read-only operation. It can be repeated without affecting the state of the resource (idempotent) and can be cached
- POST is a read-write operation and may change the state of the resource and provoke side effects on the server
  - Web browsers warn you when refreshing a page generated with POST

POST vs. PUT

What is the right way of creating resources (initialize their state)?

```
PUT /resource/{id}
```

- Problem: How to ensure resource {id} is unique?
  - (Resources can be created by multiple clients concurrently)

```
POST /resource
201 Created
Location: /resource/{id}
```

- Solution: let the server compute the unique id

Content Negotation (Conneg)

Negotiating the message format does not require to send more messages

```
GET /resource
Accept: text/html, application/xml,
   application/json
```

1. The client lists the set of format (MIME types) that it understands

```
200 OK
Content-Type: application/json
```

2. The server chooses the most appropriate one for the reply

Forced Content Negotiation

- The generic URI supports content negotiation

```
GET /resource
Accept: text/html, application/xml,
   application/json
```

- The specific URI points to a specific representation format using the postfix

```
GET /resource.html
GET /resource.xml
GET /resource.json
```

  - Warning: This is a conventional “best practice” (not a standard)

Exception Handling

Learn to use HTTP Standard Status Codes

100	Continue
200	OK
201	Created
202	Accepted
203	Non-Authoritative
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect

4xx Client's fault

400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed

500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

5xx Server's fault

Idempotent vs. Unsafe (1)

- Idempotent requests can be processed multiple times without side effects (the state of the server does not change)

```
GET /book
PUT /order/x
DELETE /order/y
```

- If something goes wrong (server down, server internal error), the request can be simply replayed until the server is back up again

Idempotent vs. Unsafe (2)

- Unsafe requests modify the state of the server and cannot be repeated without further effects:

```
Withdraw(200$) //unsafe
Deposit(200$)  //unsafe
```

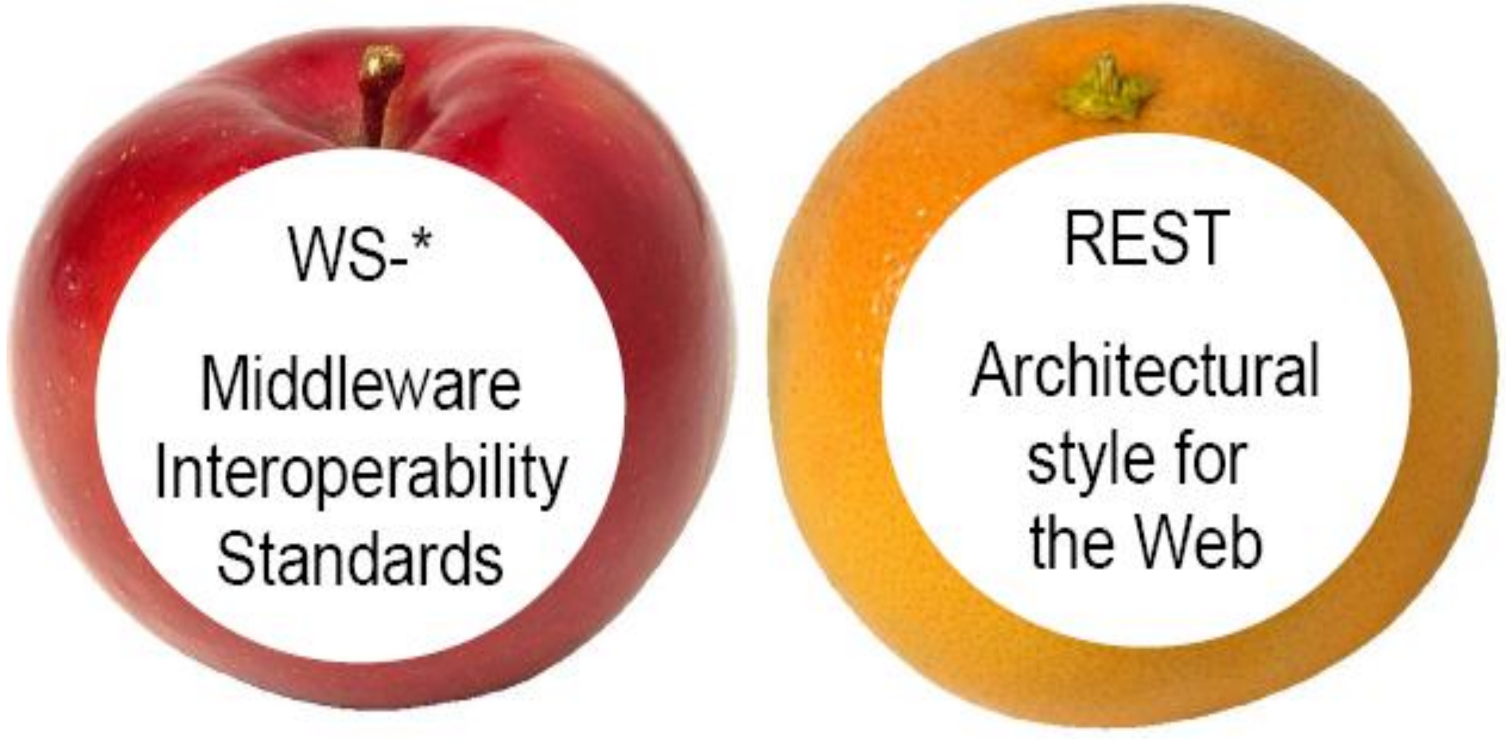
- Unsafe requests require special handling in case of exceptional situations (e.g., state reconciliation)

```
POST /order/x/payment
```

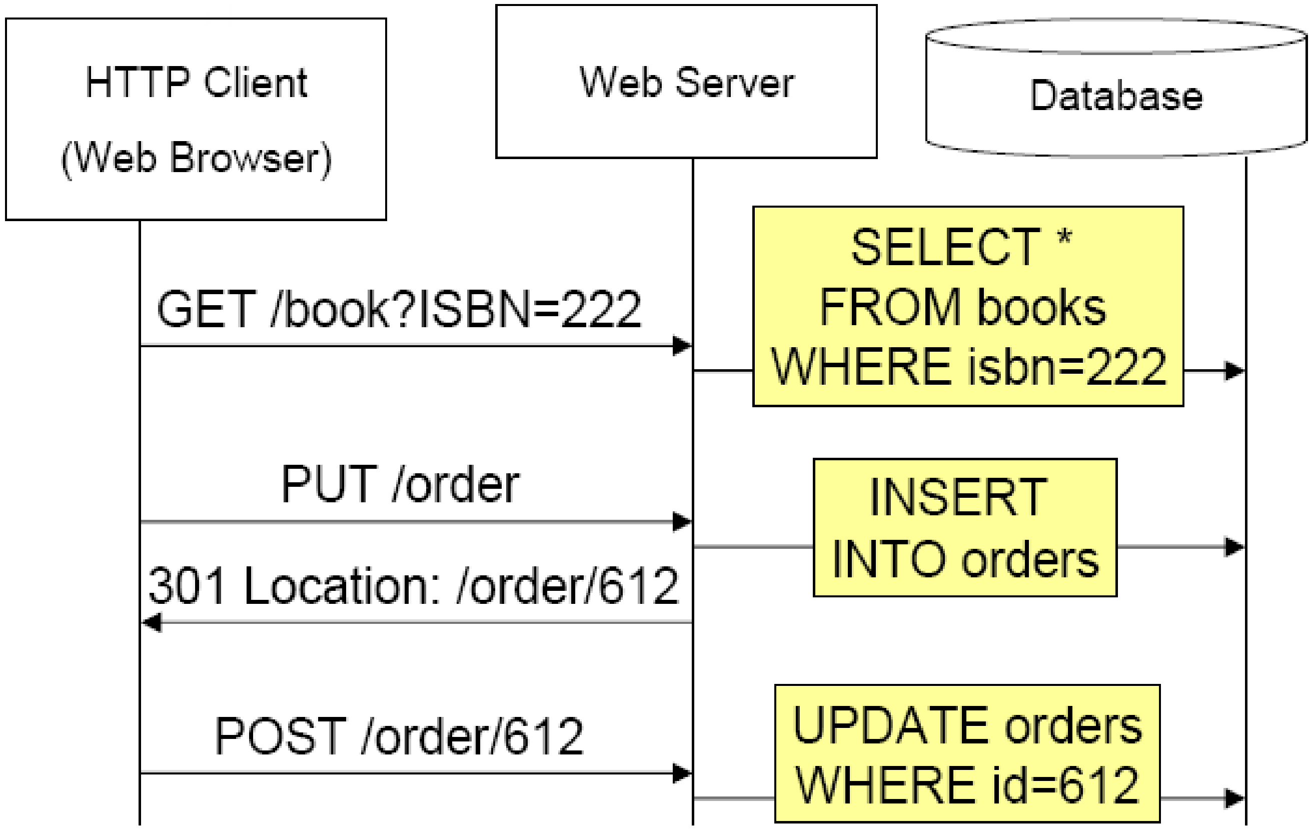
- In some cases the API can be redesigned to use idempotent operations:

```
B = GetBalance() //safe
B = B + 200$     //local
SetBalance(B)    //safe
```

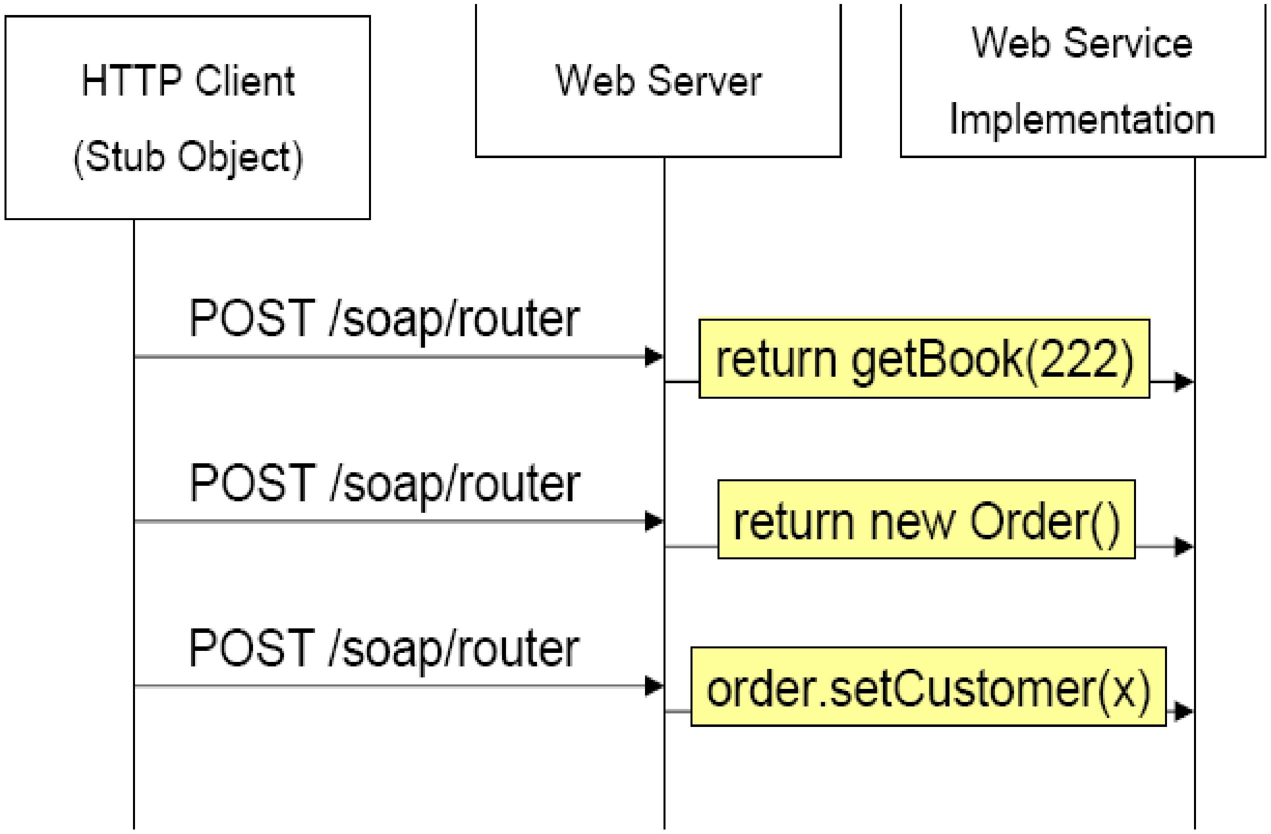
Comparing WS-* vs. REST?


WS-*
Middleware Interoperability Standards
REST
Architectural style for the Web

RESTful Web Application: example

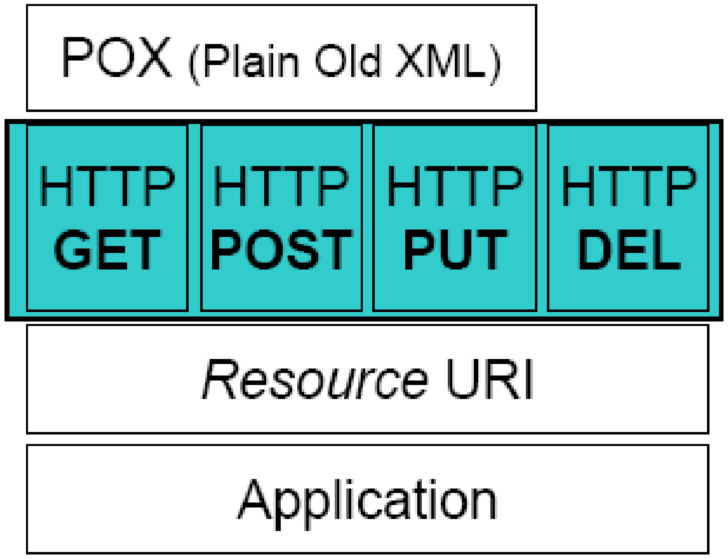


Web Service Example (from REST perspective)



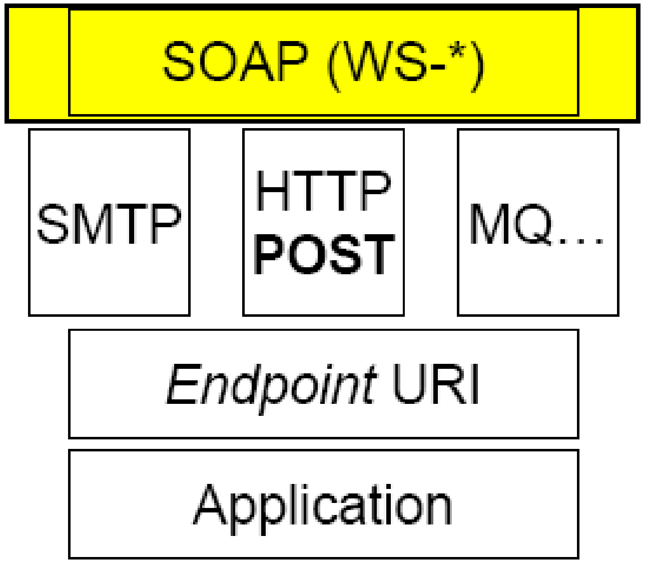
Main difference: REST vs. SOAP (1)

- “The Web is the universe of globally accessible information” (Tim Berners Lee)
  - Applications should publish their data on the Web (through URI)

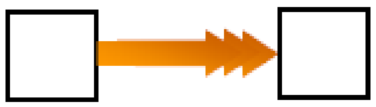


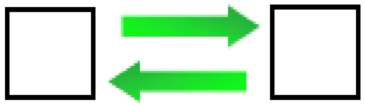
Main difference: REST vs. SOAP (2)

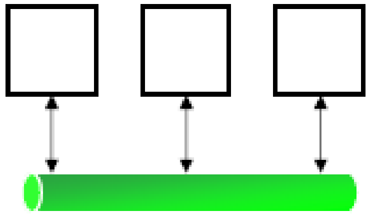
- “The Web is the universal transport for messages”
  - Applications get a chance to interact but they remain “outside of the Web”

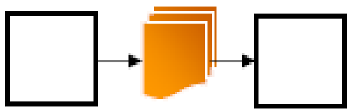


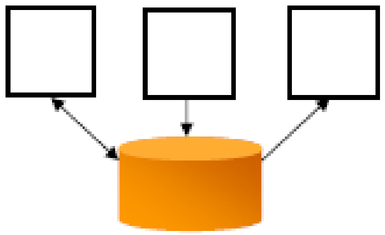
REST as a connector

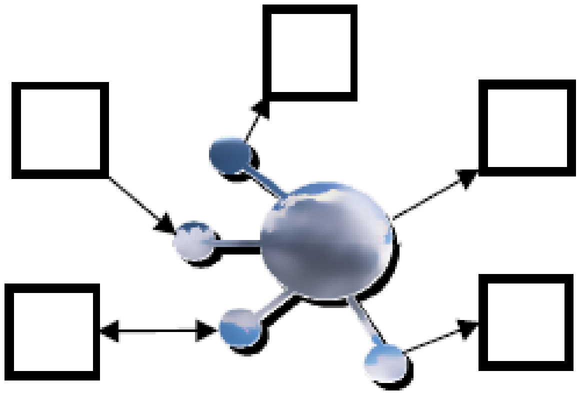

Stream


Remote Procedure Call


Message Bus
Events


File Transfer


Shared Data


REpresentational
State Transfer

Stateless or Stateful?

- REST provides explicit state transitions
  - Communication is stateless
  - Resources contain data and links representing valid state transitions
  - Clients maintain state correctly by following links in generic manner
- Techniques for adding session to HTTP:
  - Cookies (HTTP Headers)
  - URL Re-writing
  - Hidden Form Fields

What about service description?

- REST relies on human readable documentation that defines requests URIs and responses (XML, JSON)
- Interacting with the service means hours of testing and debugging URIs manually built as parameter combinations. (Is it really that simpler building URIs by hand?)
- Why do we need strongly typed SOAP messages if both sides already agree on the content?
- WADL proposed Nov. 2006
- XML Forms enough?

What about security? (1)

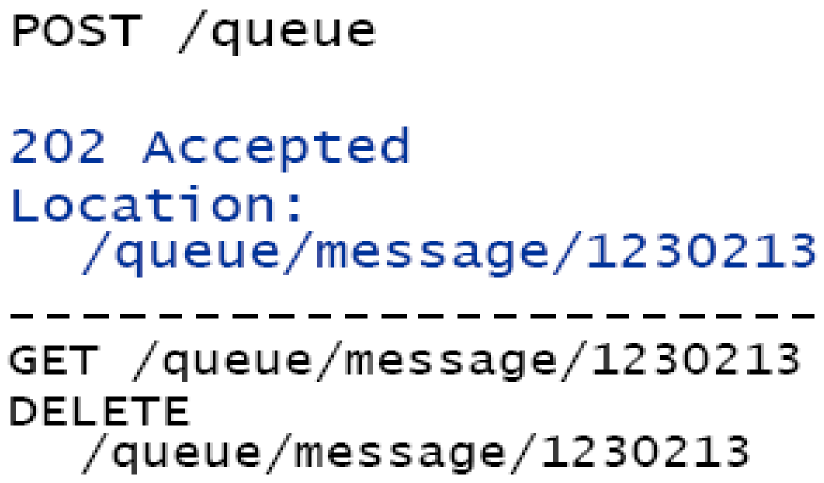
- REST security is all about HTTPS
- Proven track record (SSL1.0 from 1994)
- Secure, point to point communication
(Authentication, Integrity and Encryption)

What about security? (2)

- SOAP security extensions defined by WS-Security (from 2004)
- XML Encryption (2002)
- XML Signature (2001)
- Implementations are starting to appear now
  - Full interoperability moot
  - Performance?
- Secure, end-to-end communication – Self-protecting SOAP messages (does not require HTTPS)

What about asynchronous reliable messaging? (1)

- Although HTTP is a synchronous protocol, it can be used to “simulate” a message queue



```
POST /queue

202 Accepted
Location:
  /queue/message/1230213
------------------------
GET /queue/message/1230213
DELETE
  /queue/message/1230213
```

What about asynchronous reliable messaging? (2)

- SOAP messages can be transferred using asynchronous transport protocols and APIs (like JMS, MQ, …)
- WS-Addressing can be used to define transport-independent endpoint references
- WS-ReliableExchange defines a protocol for reliable message delivery based on SOAP headers for message identification and acknowledgement

SOAP and REST

- RESTafarians would like Web services to use and not to abuse the architecture of the Web
- Web Services more valuable when accessible from the Web
- Web more valuable when Web Services are a part of it
- W3C Workshop on Web of Services for Enterprise Computing, 27-28 February 2007 – with IBM, HP, BEA, IONA, Yahoo, Sonic, Redhat/JBoss, WSO2, Xerox, BT, Coactus Consulting, Progress Software, and others

REST and SOAP Similarities

- Existing Web applications can gracefully support both traditional Web clients (HTML/POX) and SOAP clients in a RESTful manner
- MIME Type: application/soap-xml
- SOAP with document/literal style not so different from REST (or at least HTTP/POX) - apart from the GET/POST misuse and the extra <envelope><header><body> tags in the payload

Debunking the Myth (1)

- Many “RESTafarians” have taken the position that REST and Web services are somehow incompatible with one another
- Fact: recent versions of SOAP and WSDL have been designed specifically to enable more RESTful use of Web services
- SOAP1.2
  - SOAP1.2 Response MEP
  - Web Method

Debunking the Myth (2)

- WSDL2.0
  - HTTP binding permits assigning verbs (GET, POST, etc.) on a per-operation basis
  - Attribute to mark an operation as safe (and thus cacheable)
- Unfortunately, the implementations of Web services runtimes and tooling have made RESTful use of Web services difficult

Conclusion and Outlook (1)

- Service-Oriented Architecture can be implemented in different ways
- You should generally focus on whatever architecture gets the job done and recognize the significant value of open standards but try to avoid being religious about any specific architectures or technologies
- The right steps have been taken in the development of some of the more recent WS-* specifications to enable this vision to become reality

Conclusion and Outlook (2)

- SOAP and the family of WS-* specifications have their strengths and weaknesses and will be highly suitable to some applications and positively terrible for others. Likewise with REST. The decision of which to use depends entirely on the circumstances of the application
- In the near future there will be a single scalable middleware stack, offering the best of the Web in simple scenarios, and scaling gracefully with the addition of optional extensions when more robust quality of service features are required

References

- Leonard Richardson, Sam Ruby, **RESTful Web Services**, O'Reilly, May 2007
- Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine, 2000, Chapter 5 http://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf
- **W3C Workshop on Web of Services for Enterprise Computing, 27-28 February 2007** http://www.w3.org/2007/01/wos-ec-program.html
- Sun, JSR311 - **Java API for RESTful Web Services** http://jcp.org/en/jsr/detail?id=311
- Marc J. Hadley (Sun), Web Application Description Language (**WADL**) https://wadl.dev.java.net/
- Thomas Bayer, **REST Web Services** – Eine Einfuehrung (November 2002) http://www.oio.de/public/xml/rest-webservices.pdf
- Michi Henning, **The Rise and Fall of CORBA**, Component Technologies, Vol. 4. No. 5, June 2006
- Stefan Tilkov, REST Patterns and Antipatterns, JAOO 2008
- Cesare Pautasso, Erik Wilde, **From SOA to REST**, WWW 2009 Tutorial
- Jacob Nielsen, **URI are UI**, http://www.useit.com/alertbox/990321.html
- Douglas Crockford, **JSON - the fat-free alternative to XML**, XML 2006.
- Cesare Pautasso, Olaf Zimmermann, Frank Leymann, **RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision,** 17th International World Wide Web Conference (WWW2008), Bejing,China, April 2008. http://www.jopera.org/docs/publications/2008/restws

FINE