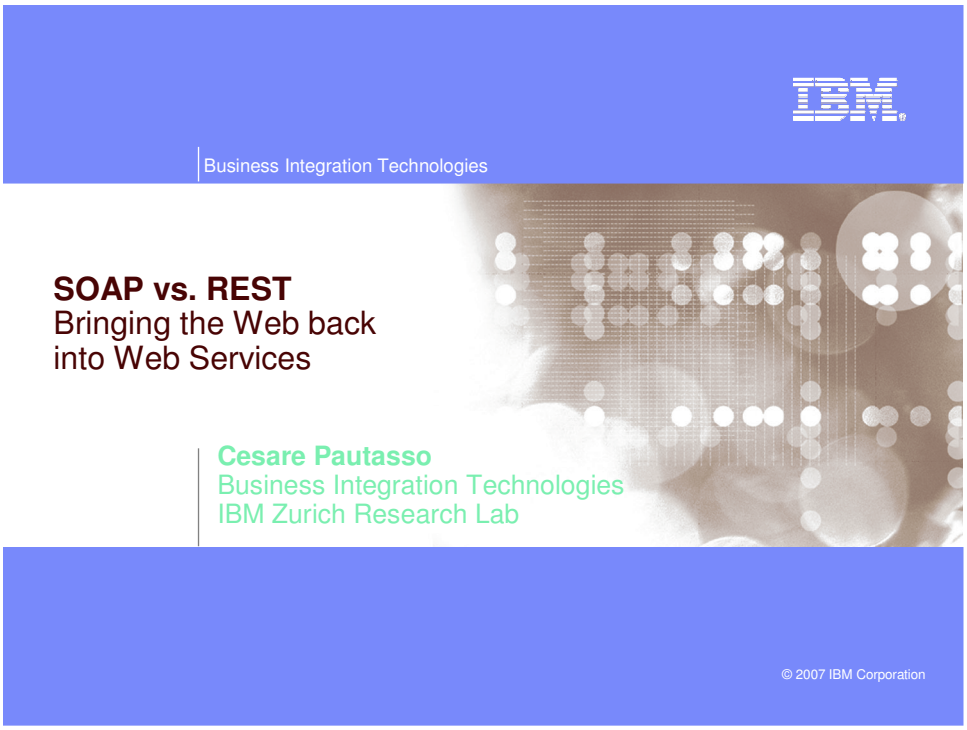


Outline

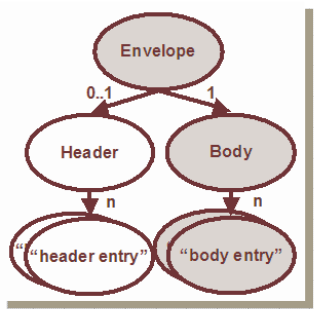
- What is SOAP?
- What is REST?
- What are they good for? Strengths and Weaknesses
- Technology Comparison
  - Arguments from each side
- Outlook
  - Bringing the Web back into Web Services

What is SOAP?

- It started in 1998 as a remote procedure call protocol that would go through firewalls (the **Simple** Object Access Protocol)
- To achieve that, the first version of SOAP defined:
  - An extensible message format based on XML (<soap:Envelope>, <soap:Body>, and a number of optional <soap:Headers>)
  - ~~A serialization mechanism to encode RPC parameters in XML (before XML Schema existed)~~
  - A binding to an existing transport protocol that would go through the firewalls (HTTP)
- In the past 7 years SOAP has evolved quite a lot to become more flexible in terms of the XML data to be transferred (**document/literal**) and also protocol agnostic (supporting different bindings of WSDL)

SOAP is just SOAP

- A protocol “framework”, to deliver the necessary interoperability between message-based middleware tools across the entire industry



SOAP message containment structure

```
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/Schema"
        xmlns:xsi="http://www.w3.org/2001/Schema-instance">
  <SOAP-ENV:Header>
    <pQP:priority
    xmlns:pQP="http://premierquotes.com/ns/priority"
    SOAP-ENV:mustUnderstand="1">
    high
    </pQP:priority>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <pQ:getPhoneNumber
    xmlns:pQ="http://premierquotes.com/ns/employees"
    SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/">
      <first-name xsi:type="xsd:string">Ed U.</first-name>
      <last-name xsi:type="xsd:string">Cate</last-name>
    </pQ:getPhoneNumber>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Header entry · Header must be processed · Method invocation · Body entry · Use SOAP encoding · Method parameters · Method parameter types · Method parameter values

SOAP message example

From: Olaf Zimmerman, OOPSLA 2005 Tutorial on Building Service-Oriented Architectures with Web Services

Where do Web services come from?

- Address the **problem of enterprise software standardization**
- Enterprise Computing Standards for Interoperability (WS started 2001)
- A layered architecture with a variety of messaging, description and discovery specifications
- Do things from the ground up, quickly, in well factored, distinct, tightly focused specifications
- Tools will hide the complexity

- REST advocates have come to believe that their ideas are just as applicable to solve application integration problems.
- Are all specifications really composable?
- “Look ma’, no tools!”

Web Services weaknesses

- High perceived complexity
- Problematic standardization process
  - Infighting
  - Lack of architectural coherence
  - Fragmentation
  - Design by committee
  - Feature Bloat (Merge of competing specs)
  - Lack of reference implementations
  - Standardization of standards (WS-I)

WS-PageCount	
Messaging	232 pages
Metadata	111 pages
Security	230 pages
WS-BPEL	195 pages
XML/XSD	599 pages
Transactions	39 pages

Numbers from Tim Bray

- Is this starting to look like CORBA?
- When is Web services interoperability start to really work?
- Do we really have to buy XML appliances to get good performance?

What are Web services good for?

- **SOAP/WSDL as the gateway technology** to enable interoperability, common ground for messaging applications that work both over HTTP and other protocols
- Legacy reuse and integration
  - Pre-existing systems are not built to be Web-friendly – they use non-HTTP protocols, multicast, RPC/RMI binary protocols, asynchronous messaging, batched bulk transfers
- Flexibility and adaptation
  - The same interface can be easily bound to different protocols as business and technological requirements change

REST

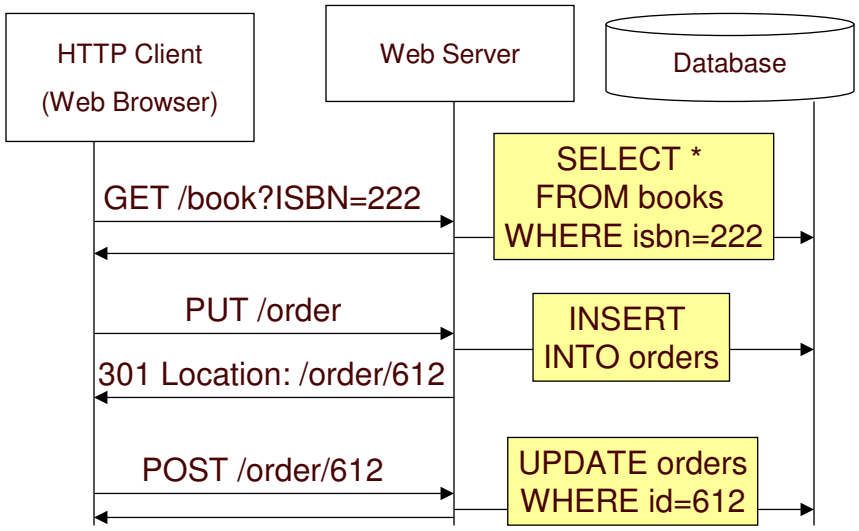
Monsieur Jourdain,
Le Bourgeois Gentilhomme
Molière, 1670

What is REST? (REpresentational State Transfer)

- REST is the architecture of the Web, its principles have been used to explain why the HTTP protocol scales so well (Roy Fielding PhD) (REST 1994, HTTP 1.0 1989)

1. Resource Identification through URI
2. **Uniform Interface** for all resources:
   - GET (Query the state, idempotent, can be cached)
   - POST (Modify, transfer the state)
   - PUT (Create a resource)
   - DELETE (Delete a resource)
3. "Self-Descriptive" Messages (Format/compression can be negotiated)
4. Hyperlinks between different media types

RESTful Web Application Example

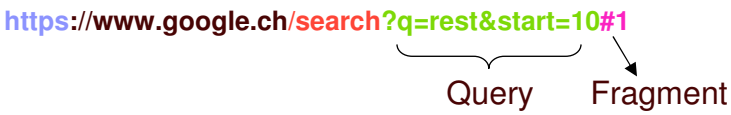


Uniform Interface Principle (CRUD Example)

	SQL	REST
CREATE	Insert	PUT
READ	Select	GET
UPDATE	Update	POST
DELETE	Delete/Drop	DELETE

URI: Uniform Resource Identifier

- Internet Standard for resource naming and identification (originally from 1994, revised until 2005)
- Examples:



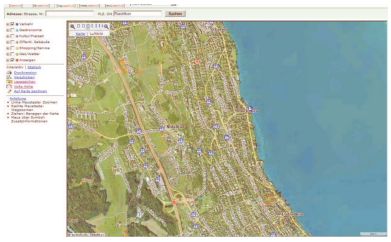
- RESTafarians advocate the use of “nice” URIs
- In most HTTP stacks URIs cannot have arbitrary length (4Kb)

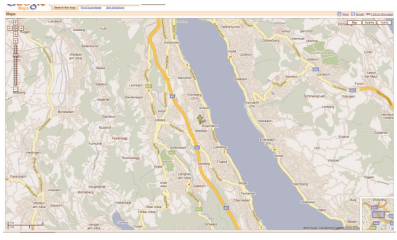
What is a nice URI?

Prefer Nouns to Verbs

Keep them Short

http://map.search.ch/rueschlikon





http://maps.google.com/maps?f=q&hl=en&q=rueschlikon,+switzerland&layer=&ie=UTF8&z=12&om=1&iwloc=addr

High REST vs. Low REST

- Best practices differ:

Low REST

- HTTP GET for idempotent requests, POST for everything else
- Responses in any MIME Type (e.g., XHTML)

High REST

- Usage of nice URIs recommended
- Full use of the 4 verbs: GET, POST, PUT, and DELETE (*)
- Responses using Plain Old XML (**)

(*) Some firewalls or HTTP Proxies may drop PUT/DELETE requests

(**) POX could be replaced with RDF, JSON, YAML, or ATOM (highly debated)

REST Strengths

- Simplicity
  - Uniform interface is **immutable** (no problem of breaking clients)
- HTTP/POX is ubiquitous (goes through firewalls)
- Stateless/Synchronous interaction
- Proven scalability
  - "after all the Web works", **caching,** clustered server farms for QoS
- Perceived ease of adoption (light infrastructure)
  - just need a browser to get started - no need to buy WS-* middleware
- Grassroots approach
- Leveraged by all major Web 2.0 applications
  - 85% clients prefer Amazon RESTful API (*)
  - Google recently announced it would no longer support its SOAP/WSDL API

(*) http://www.oreillynet.com/pub/wlg/3005

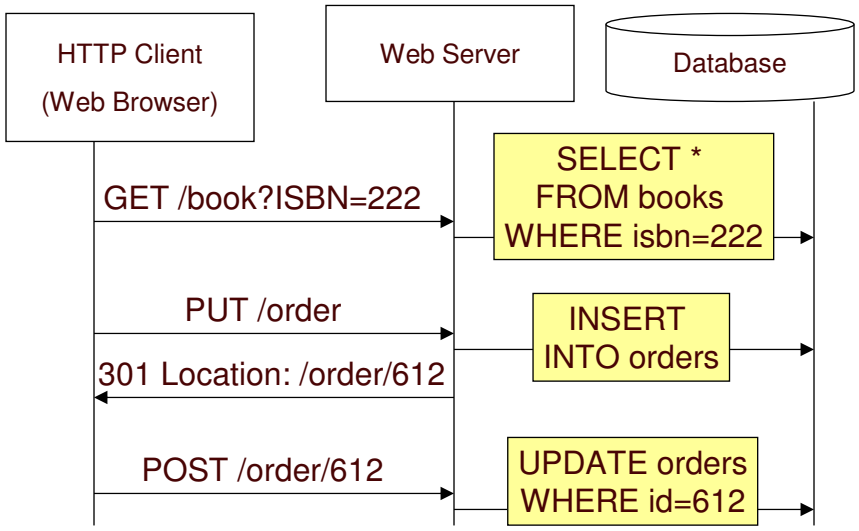
REST Weaknesses

- Confusion (high REST vs. low REST)
  - Is it really 4 verbs? (HTTP 1.1. has 8 verbs: HEAD, **GET, POST**, PUT, DELETE, TRACE, OPTIONS, and CONNECT)
- Mapping REST-style synchronous semantics on top of back end systems creates design mismatches (when they are based on asynchronous messaging or event driven interaction)
- Cannot deliver enterprise-style "-ilities" beyond HTTP/SSL
- Challenging to identify and locate resources appropriately in all applications
- Apparent lack of standards (other than URI, HTTP, XML, MIME, HTML)
- Semantics/Syntax description very informal (user/human oriented)

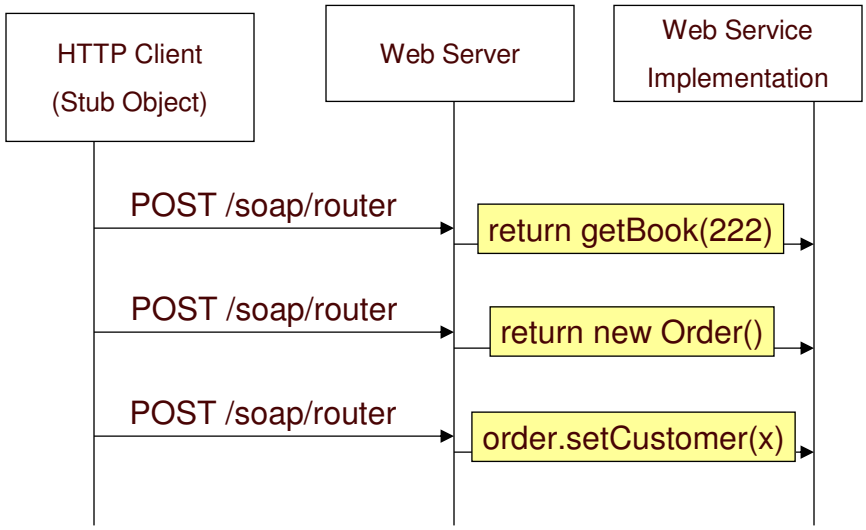
Business Integration Technologies

SOAP vs. REST Comparison

RESTful Web Application Example

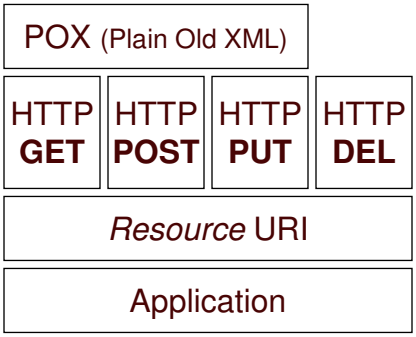


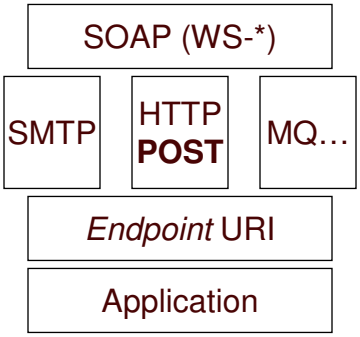
Web Service Example (from REST perspective)



Main difference: REST vs. SOAP

- "The Web is the universe of globally accessible information" (Tim Berners Lee)
  - Applications should publish their data on the Web (through URI)
- "The Web is the universal transport for messages"
  - Applications get a chance to interact but they remain "outside of the Web"





REST vs. SOAP – Conceptual Comparison

The Web as an open publishing medium	The Web as a cross enterprise communication medium
Resource Documents originally meant for human consumption	Business Documents for business process driven consumption
Resource oriented – access informational resources	Activity/Service oriented – actions that may be performed orthogonally to the resources upon which they act
Services for outsourcing	Services for cross-enterprise trading
The more clients the better	Selected set of clients (visibility and access controlled by contracts)

REST vs. SOAP – Technology Comparison

User-driven interactions via forms	Orchestrated reliable event flows
Few operations (generic interface) on many resources	Many operations (service interface) on few resources
URI: Consistent naming mechanism for resources	Lack of standard naming mechanism
Focus on scalability and performance of large scale distributed hypermedia systems	Focus on design of integrated (distributed) applications
Composition with Mashups	Composition via Business Processes

REST vs. SOAP – Protocol Comparison

XML in – XML out (with POST)	SOAP in – SOAP out (with POST)
URI in – XML out (with GET)	
"Self-Describing" XML	Strong Typing (XML Schema)
HTTP only	"Transport independent"
HTTP/SSL is enough – no need for more standards	Heterogeneity in QoS needs. Different protocols may be used
HTTP is an application protocol	HTTP as a transport protocol
Synchronous	Synchronous and Asynchronous
Do-it-yourself when it comes to "reliable message delivery", "distributed transactions"	Foundation for the whole WS* advanced protocol stack

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 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?

- Client stubs can be built from WSDL descriptions in most programming languages
- Strong typing
- Each service publishes its own interface with different semantics
- WSDL 1.1 (entire port type can be bound to HTTP GET or HTTP POST or SOAP/HTTP POST or other protocols)
- WSDL 2.0 (more flexible, each operation can choose whether to use GET or POST)

What about state management?

- REST provides explicit state transitions
  - Server 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

- SOAP services have implicit state transitions
  - Servers may maintain conversation state across multiple message exchanges
  - Messages contain only data (but do not include information about valid state transitions)
  - Clients maintain state by guessing the state machine of the service
- Techniques for adding session to SOAP:
  - Session Headers (non standard)

(*) Each client request to the server must contain all information needed to understand the request, without referring to any stored context on the server.

What about security?

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

- 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)

REST vs. SOAP Design Methodology

1. Identify resources to be exposed as services (e.g., book catalog, purchase order)
2. Define “nice” URLs to address them
3. Distinguish read-only and side-effects free resources (GET) from modifiable resources (POST, PUT, DELETE)
4. Relationships (e.g., containment, reference) between resources correspond to hyperlinks that can be followed to get more details
5. Implement and deploy on Web server

1. List what are the service operations (the “verbs”) into the service’s WSDL document
2. Define a data model for the content of the messages exchanged by the service (XML Schema data types)
3. Choose an appropriate transport protocol to bind the operation messages and define the corresponding QoS, security, transactional policies
4. Implement and deploy on the Web service container (note the corresponding SOAP engine end-point)

More Challenges for REST (and plain SOAP)

- Reuse: service granularity, discovery
- Interface Contracts (many artifacts, not just WSDL)
- Heterogeneity
  - Communication style (synchronous, asynchronous, bulk transfer, flow control, reliable message delivery, non HTTP protocols)
  - Legacy wrapping (back end integration)
- Transactions
- Asynchronous Event Notification
- Change Management (versioning, maintenance)
- Complex message/document flows (intermediaries, routing)
- *Can you run a BPEL process over RESTful Web services?*

How many views of the world?

- Resources and State
- Files and Hot Folders
- Continuous Data Feeds
- Services and Invocations
- Screenscraping Mainframes
- Messages, Events and Endpoints
- Operating system shell commands
- Jobs submitted to a Cluster/Grid scheduler

Putting the Web back into Web services

- 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

- XML
- XML Schema
- HTTP
- Loose coupling important for scalability

- O/XML binding problematic
- Alternatives to XML starting to appear (e.g., JSON, YAML, RDF)
- Data contract needs to be defined even with uniform interface
- Although REST claims SOAP misuses the verbs
- Interpretations differ

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

- 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
- 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

- 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.
- 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.
- The right steps have been taken in the development of some of the more recent WS-* specifications to enable this vision to become reality

References

- 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 Microsystems,
  Web Application Description Language (**WADL**)
  https://wadl.dev.java.net/wadl20061109.pdf
- Thomas Bayer, Orientation in Objects GmbH,
  REST Web Services – Eine Einfuehrung (November 2002)
  http://www.oio.de/public/xml/rest-webservices.pdf
- Michi Henning, ZeroC, **The Rise and Fall of CORBA**, Component Technologies, Vol. 4. No. 5, June 2006
- Jacob Nielsen, **URI are UI**, http://www.useit.com/alertbox/990321.html

Business Integration Technologies

SOAP vs. REST
Bringing the Web back into Web Services

Cesare Pautasso
Business Integration Technologies
IBM Zurich Research Lab